

IMAGE PROCESSING

- **Digital image processing** is the use of a digital computer to process digital images through an algorithm
- The capture of images with devices such as flatbed scanners and digital cameras
- The representation and storage of images in efficient file formats
- Constructing the algorithms in image-manipulation programs such as Adobe Photoshop



DIGITAL IMAGES

- Input devices:
 - scanners
 - cameras
 - camcorders
- Output devices:
 - display screens
 - printers
- Processing:
 - file compression
 - various transformations



TRANSFORMATIONS

Convert from color to grayscale

Rotate an image

Apply color filtering to an image



TRANSFORMATIONS

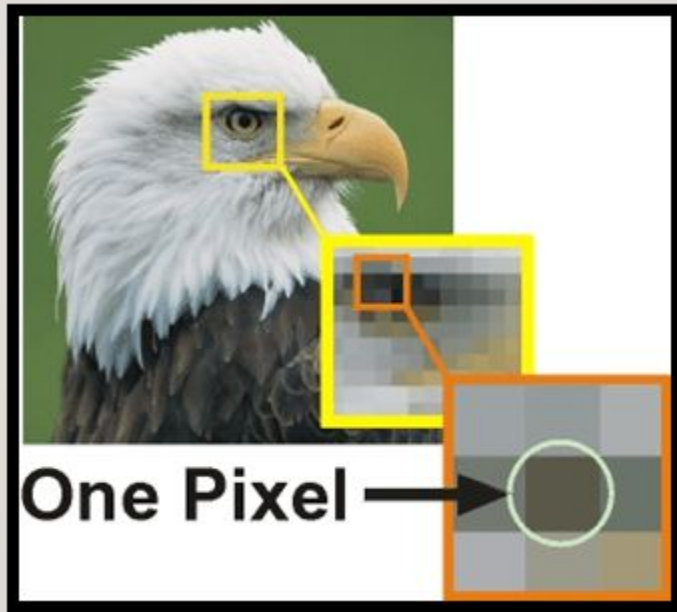
- Highlight a particular area in an image
- Blur all or part of an image
- Sharpen all or part of an image
- Control the brightness of an image
- Perform edge detection on an image
- Enlarge or reduce an image's size
- Apply color inversion to an image
- Morph an image into another image

REPRESENTING IMAGES

- An image can be represented as a two-dimensional grid of RGB values (pixels)
- To capture an image (via camera or scanner), a continuous range of color info is sampled as a set of discrete color values
- All processing works with the grid of RGB values
- Output maps the grid to a file, display, or printer



PIXEL



RGB SYSTEMS

- 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.
- The size of a pixel is determined by the size and resolution of the display.
- Setting the resolution to smaller values increases the size of the pixels, making the lines on the screen appear more ragged.



RGB SYSTEMS

- Each pixel represents a color.
- Among the various schemes for representing colors, the RGB system is a common one.
- The letters stand for the color components of red, green, and blue, to which the human retina is sensitive.
- These components are mixed together to form a unique color value.
- Naturally, the computer represents these values as integers, and the display hardware translates this information to the colors you see. Each color component can range from 0 through 255.
- The value 255 represents the maximum saturation of a given color component, whereas the value 0 represents the total absence of that component.

RGB SYSTEMS

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)

ANALOG AND DIGITAL INFORMATION

- Representing photographic images in a computer poses an interesting problem.
- As you have seen, computers must use digital information which consists of discrete values, such as individual integers, characters of text, or bits in a bit string.
- However, the information contained in images, sound, and much of the rest of the physical world is analog.
- Analog information contains a continuous range of values

SAMPLING

- Sampling devices measure discrete color values at distinct points on a two-dimensional grid.
- To create an image which is digital, we need to covert continuous data into digital form. There are two steps in which it is done.
- Sampling
- Quantization

-
- Since an image is continuous not just in its co-ordinates (x axis), but also in its amplitude (y axis), so the part that deals with the digitizing of co-ordinates is known as sampling.
 - And the part that deals with digitizing the amplitude is known as quantization.

IMAGE FILE FORMAT

- Once an image has been sampled, it can be stored in one of many 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).

DATA-COMPRESSION SCHEMES

- Various data-compression schemes are used to reduce the file size of a JPEG image.
- One scheme examines the colors of each pixel's neighbors in the grid. 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.

-
- To save even more bits, 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

THE IMAGES MODULE

- A non-standard, open source module that includes a set of classes and methods for processing images
- Can edit scripts in IDLE, then run them from the shell or a terminal prompt

```
python3 myprogram.py
```


-
- 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 $(\text{width} - 1, \text{height} - 1)$ 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

-
- The **images** module includes a class named **Image**.
 - Image class represents an image as a two-dimensional grid of RGB values.
 - The methods for the Image class are listed in Table 7-4.
 - In this table, the variable *i* refers to an instance of the Image class.

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

Table 7-4 The **Image** methods

A SIMPLE SESSION

```
>>> from images import Image
```


A SIMPLE SESSION

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

The image file must be in the
current working directory

A SIMPLE SESSION

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



draw must be run to display the image

A SIMPLE SESSION

```
>>> from images import Image  
  
>>> image = Image('smokey.gif')  
  
>>> image.draw()  
  
>>> image.getWidth(), image.getHeight()  
(300, 225)
```



The comma creates a tuple of results

A SIMPLE SESSION

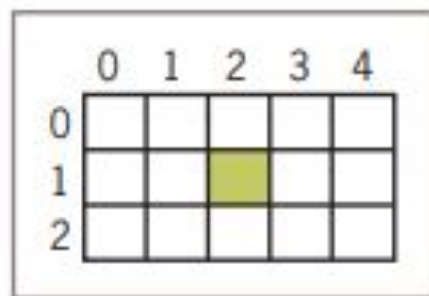
```
>>> from images import Image  
  
>>> image = Image('smokey.gif')  
  
>>> image.draw()  
  
>>> image.getWidth(), image.getHeight()  
(300, 225)  
  
>>> image.getPixel(0, 0)  
(206, 224, 122)
```



Screen coordinates: the origin (0, 0) is the upper left corner
y-values increase as you move down

TRAVERSING 2 DIMENSIONAL GRID

- Many image-processing algorithms use a nested loop structure to traverse a two-dimensional grid of pixels.
- Figure shows such a grid.
- Its height is 3 rows, numbered 0 through 2.
- Its width is 5 columns, numbered 0 through 4.
- Each data value in the grid is accessed with a pair of coordinates using the form (<column>,<row>).
- Thus, the datum in the middle of the grid, which is shaded, is at position (2, 1). The datum in the upper-left corner is at the origin of the grid, (0, 0)



	0	1	2	3	4
0					
1					
2					

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

PROCESSING A 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)
```

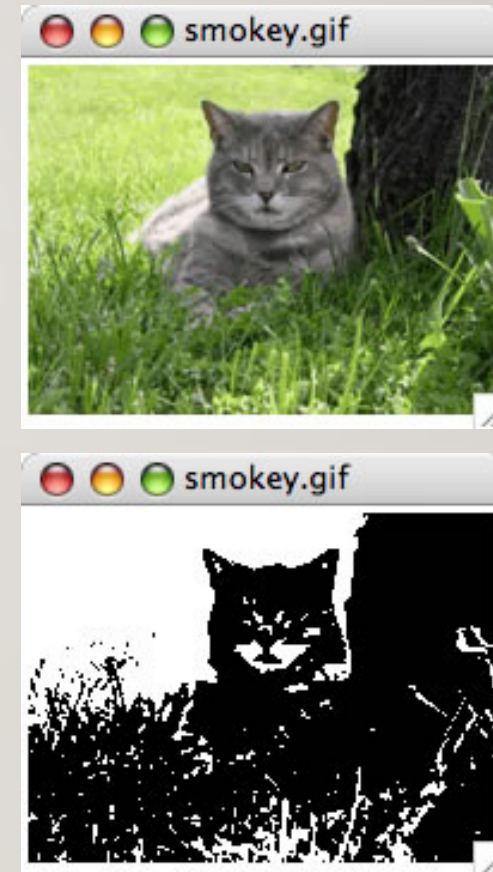
2 rows,
3 columns

The next code segment uses a nested **for** loop to fill a blank image in red:

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

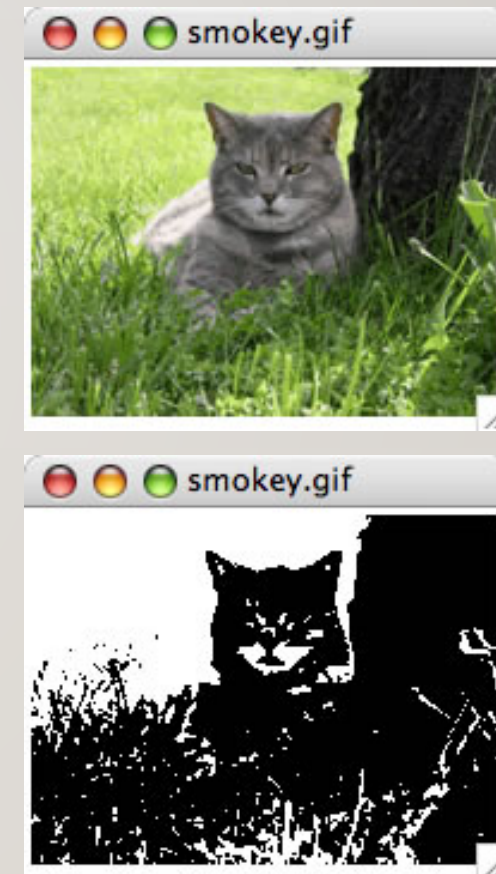

TRANSFORMATIONS: BLACK AND WHITE

- Compute the average of the three color components in a pixel
- If the average is less than 128, then set the pixel's three color components to 0 (black)
- Otherwise, set them to 255 (white)



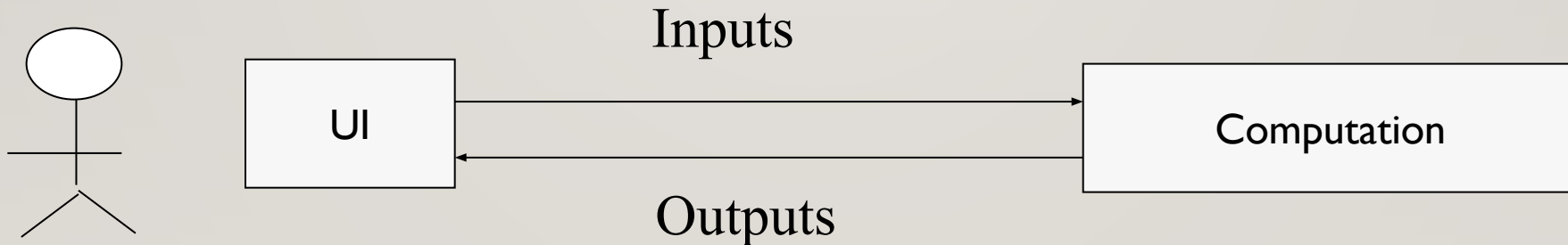
TRANSFORMATIONS: BLACK AND WHITE

```
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)
```



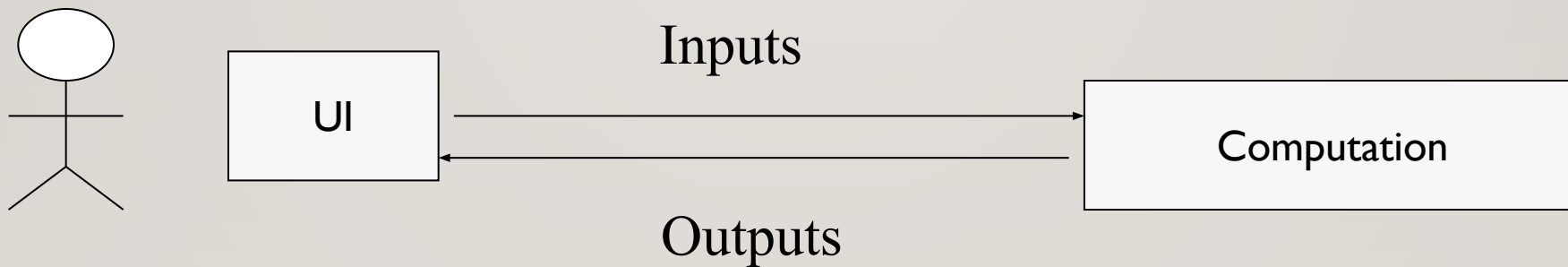
WHAT IS A USER INTERFACE?

- A set of hardware devices (touch screen, monitor, keyboard, mouse, microphone, speakers)
- Software (input/output functions)
- Allows human beings to use a computer effectively



TYPES OF USER INTERFACES

- GUI (graphical user interface)
- TUI (terminal-based user interface)



TERMINAL-BASED USER INTERFACE (TUI)

Supports input via the keyboard and output via the monitor

In Python, the I/O functions are **input** and **print**

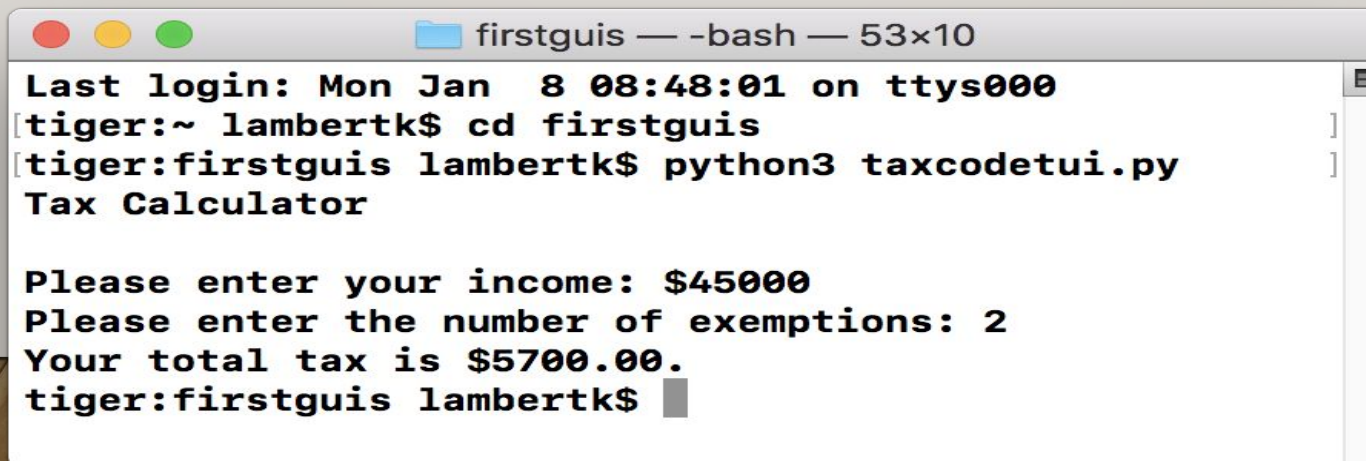
```
print("Tax Calculator\n")
RATE = .15
EXEMPTION_AMOUNT = 3500.0
income = float(input("Please enter your income: $"))
exemptions = int(input("Please enter the number of exemptions: "))
tax = max(0.0, (income - exemptions * EXEMPTION_AMOUNT) * RATE)
print("Your total tax is $%0.2f." % tax)
```

TERMINAL-BASED USER INTERFACE (TUI)

Supports input via the keyboard and output via the monitor

In Python, the I/O functions are **input** and **print**

```
print("Tax Calculator\n")
RATE = .15
EXEMPTION_AMOUNT = 3500.0
income = float(input("Please enter your income: $"))
exemptions = int(input("Please enter the number of exemptions: "))
tax = max(0.0, (income - exemptions * EXEMPTION_AMOUNT) * RATE)
print("Your total tax is $%.2f." % tax)
```

A terminal window titled 'firstguis — -bash — 53x10' is shown. It displays the output of the tax calculator program. The user has entered an income of \$45000 and 2 exemptions. The program calculates a total tax of \$5700.00. The terminal text is as follows:

```
Last login: Mon Jan  8 08:48:01 on ttys000
[tiger:~ lambertk$ cd firstguis
[tiger:firstguis lambertk$ python3 taxcodetui.py
Tax Calculator

Please enter your income: $45000
Please enter the number of exemptions: 2
Your total tax is $5700.00.
tiger:firstguis lambertk$
```

PROBLEMS WITH A TUI

- Must enter inputs in a certain order
- Cannot back up to correct input mistakes or change one's mind
- Must re-enter all inputs to change just one
- I/O restricted to text

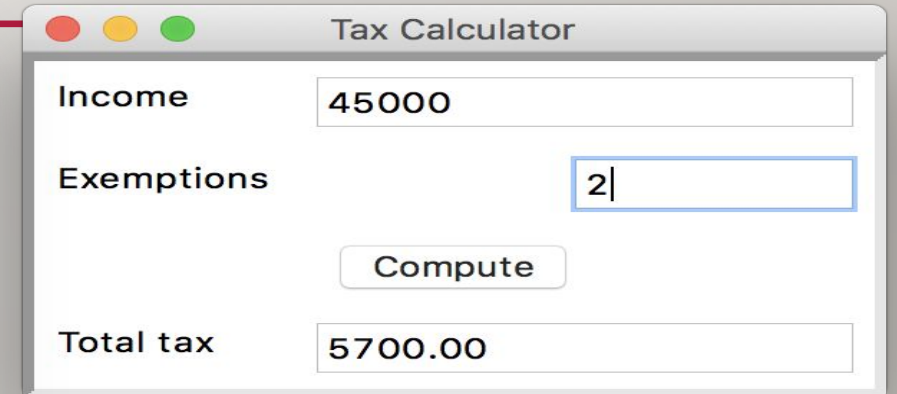
GRAPHICAL USER INTERFACE (GUI)

- Supports input via the keyboard
- Can output text and also graphical shapes representing desktop elements, such as windows, command buttons, data fields, and drop-down menus (also called “widgets”)
- Supports direct manipulation of desktop elements via the mouse or touchscreen

TUI VS GUI

```
firstguis — -bash — 53x10
Last login: Mon Jan  8 08:48:01 on ttys000
[tiger:~ lambertk$ cd firstguis
[tiger:firstguis lambertk$ python3 taxcodetui.py
Tax Calculator

Please enter your income: $45000
Please enter the number of exemptions: 2
Your total tax is $5700.00.
tiger:firstguis lambertk$
```



The GUI window titled "Tax Calculator" features a clean, modern design. It includes three input fields: "Income" with the value "45000", "Exemptions" with the value "2", and "Total tax" with the value "5700.00". A "Compute" button is positioned between the "Exemptions" and "Total tax" fields. The window has standard macOS-style window controls (red, yellow, green buttons) in the top-left corner.

Non-programmers (the 99%) do not use a TUI, they use a GUI

Only programmers (the 1%) use a TUI (and also a GUI)

Most beginning programmers program to a TUI, not a GUI

MODELS OF COMPUTATION

TUI

1. Obtain user inputs
2. Perform computations
3. Print results

GUI

1. Layout and pop up the window
2. Wait for user events
3. Handle a user event
4. Goto step 2

The model of computation for a GUI program is more complex than that of a TUI program

PROGRAMMING A GUI

- Most modern programming languages (like Python and Java) include packages or modules for programming a GUI
- In Python, this module is called **`tkinter`**

GUI RESOURCES IN PYTHON

tkinter

<http://docs.python.org/py3k/library/tkinter.html#module-tkinter>

breezypythongui

<http://home.wlu.edu/~lambertk/breezypythongui/index.html>

WHAT IS BREEZYPYTHONGUI?

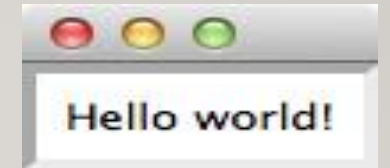
- A module of classes and functions that makes GUI programming with `tkinter` easy for beginners
- The module is free and open-source
- A tutorial and sample programs are available at the `breezypythongui` Web site

EVENT DRIVEN PROGRAMMING

- Rather than guide the user through a series of prompts, a GUI-based program opens a window and waits for the user to manipulate window components with the mouse.
- These user-generated events, such as mouse clicks, trigger operations in the program to respond by pulling in inputs, processing them, and displaying results.
- This type of software system is event-driven, and the type of programming used to create it is called **event-driven programming**.

A FIRST GUI PROGRAM: HELLO WORLD

```
from breezypythongui import EasyFrame
```

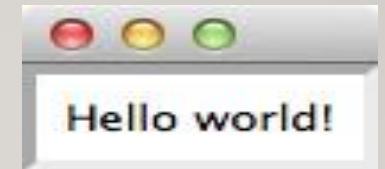


Import the abstract
window class

A FIRST GUI PROGRAM: HELLO WORLD

```
from breezypythongui import EasyFrame

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



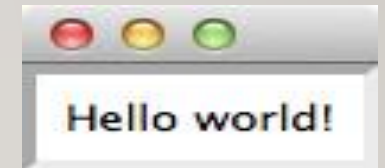
Define a subclass
to specialize it for
this application

A FIRST GUI PROGRAM: HELLO WORLD

```
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)
```



Lay out the
window and its
widgets

A FIRST GUI PROGRAM: HELLO WORLD

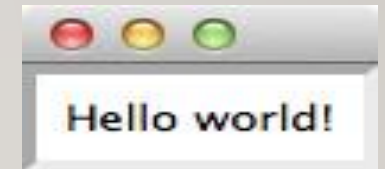
```
from breezypythongui import EasyFrame

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)

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

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



Create and launch
the application
window

CNTD:

- In all of our GUI-based programs, a new window class extends the EasyFrame class
- The EasyFrame class provides the basic functionality for any window, such as the command buttons in the title bar
- 1. Import the EasyFrame class from the breezypythongui module. This class is a subclass of tkinter's Frame class, which represents a top-level window
- 2. Define the LabelDemo class as a subclass of EasyFrame. The LabelDemo class describes the window's layout and functionality for this application

CNTD:

3. Define an `__init__` method in the `LabelDemo` class. This method is automatically run when the window is created.

- The `__init__` method runs a method with the same name on the `EasyFrame` class and then sets up any window components to display in the window.
- In this case, the `addLabel` method is run on the window itself.
- The `addLabel` method creates a window component, a label object with the text “Hello world!,” and adds it to the window at the grid position (0, 0).

CNTD:

4. The last five lines of code define a main function and check to see if the Python code file is being run as a program.

- If this is true, the main function is called to create an instance of the LabelDemo class. The mainloop method is then run on this object.
- At this point, the window pops up for viewing. Note that mainloop, as the name implies, enters a loop.
- The Python Virtual Machine runs this loop behind the scenes. Its purpose is to wait for user events, as mentioned earlier. The loop terminates when the user clicks the window's close box

THE STRUCTURE OF ANY GUI PROGRAM

```
from breezypythongui import EasyFrame

class <class name>(EasyFrame):

    def __init__(self):
        EasyFrame.__init__(self <optional args>)
        <code to set up widgets>

        <code to define event-handling methods>

# Instantiates and pops up the window.
def main():
    ApplicationName().mainloop()

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

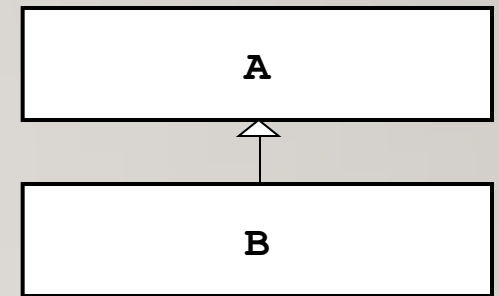
THE STRUCTURE OF ANY GUI PROGRAM

- A GUI application window is always represented as a class that extends `EasyFrame`.
- The `__init__` method initializes the window by setting its attributes and populating it with the appropriate GUI components.
- In our example, Python runs this method automatically when the constructor function `LabelDemo` is called.
- The last lines of code, beginning with the definition of the main function, create an instance of the application window class and run the `mainloop` method on this instance.
- The window then pops up and waits for user events. Pressing the window's close button will quit the program normally.

CLASSES AND INHERITANCE

- **Inheritance:** Class **B** inherits the attributes and operations of class **A**, and then adds some of its own; clients can run **A**'s methods as well as **B**'s methods

↑ = extends



```
from <some module> import <parent class name>

class <class name>(<parent class name>):
```


CLASSES AND INHERITANCE

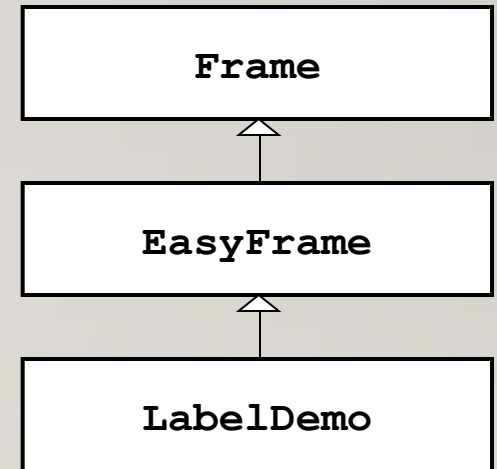
- It's easiest to build a new class by extending an existing class

In `tkinter` module

In `breezypythongui` module

In `labeldemo` module

```
from breezypythongui import EasyFrame  
  
class LabelDemo(EasyFrame):
```



METHOD HEADER

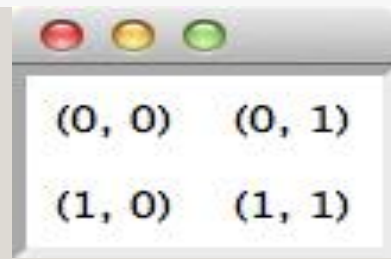
- A method header looks very much like a function header, but a method always has at least one parameter, in the first position, named `self`.
- At call time, the PVM automatically assigns to this parameter a reference to the object on which the method is called; thus, you do not pass this object as an explicit argument at call time. For example, given the method header

```
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

WINDOW LAYOUT

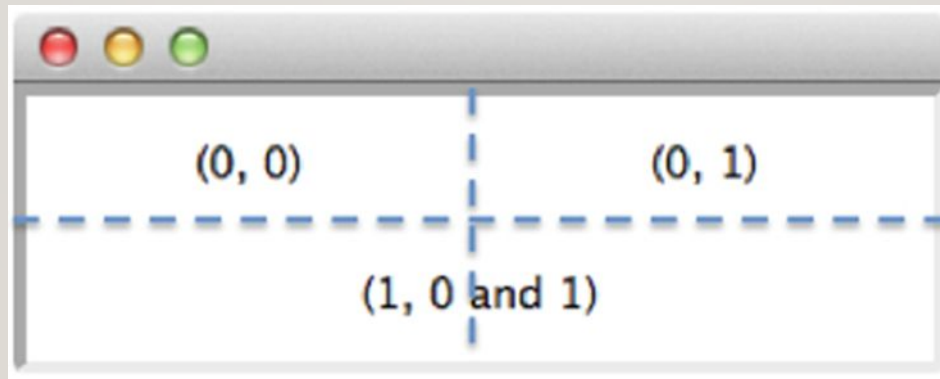
```
class LayoutDemo(EasyFrame):  
    """Displays labels in the quadrants."""  
  
    def __init__(self):  
        """Sets up the window and the labels."""  
        EasyFrame.__init__(self)  
        self.addLabel(text = "(0, 0)", row = 0, column = 0)  
        self.addLabel(text = "(0, 1)", row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



-
- Each type of window component has a default alignment within its grid position.
 - Because labels frequently appear to the left of data entry fields, their default alignment is northwest.
 - The programmer 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.
 - The programmer can force a horizontal and/ or vertical spanning of grid positions by supplying the rowspan and colspan keyword arguments when adding a component (like merging cells in a table or spreadsheet)

WINDOW ALIGNMENT AND SPANNING

```
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", colspan = 2)
```

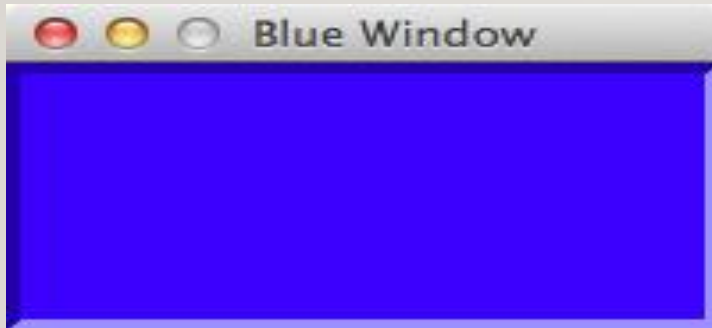


WINDOW ATTRIBUTES

- A window has several attributes. The most important ones are its
 - title (an empty string by default)
 - width and height in pixels
 - resizable (true by default)
 - background color (white by default)

OPTIONAL ARGUMENTS TO `__init__`

```
def __init__(self):  
    """Sets up the window."""  
    EasyFrame.__init__(self, title = "Blue Window",  
                        width = 200, height = 100,  
                        background = "blue",  
                        resizable = False)
```



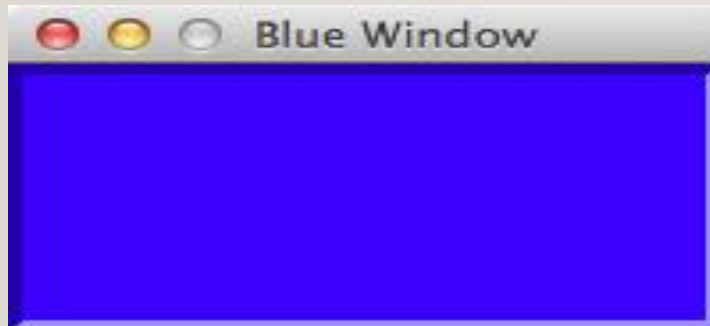
METHODS TO CHANGE WINDOW ATTRIBUTES

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

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

METHOD CALLS ON SELF (THE WINDOW)

```
def __init__(self):  
    """Sets up the window."""  
    EasyFrame.__init__(self)  
    self.setTitle("Blue Window")  
    self.setSize(200, 100)  
    self.setBackground("blue")  
    self.setResizable(False)
```



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.
- The breezypythongui module includes methods for adding each type of window component to a window.
- Each such method uses the form **self.addComponentType()**

TYPES OF WINDOW COMPONENTS AND THEIR ATTRIBUTES

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.

TYPES OF WINDOW COMPONENTS AND THEIR ATTRIBUTES

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.

I. A CAPTIONED IMAGE



A CAPTIONED IMAGE

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

A CAPTIONED IMAGE

```
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 widgets."""
        EasyFrame.__init__(self, title = "Image Demo", resizable = False)
        imageLabel = self.addLabel(text = "",
                                   row = 0, column = 0,
                                   sticky = "NSEW")
        textLabel = self.addLabel(text = "Smokey the cat",
                                   row = 1, column = 0,
                                   sticky = "NSEW")
```

A CAPTIONED IMAGE

```
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 widgets."""
        EasyFrame.__init__(self, title = "Image Demo", resizable = 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
```


A CAPTIONED IMAGE

```
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 widgets."""
        EasyFrame.__init__(self, title = "Image Demo", resizable = 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"
```

TKINTER.LABEL ATTRIBUTES

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

2. COMMAND BUTTON

- A command button is added to a window just like a label, by specifying its text and position in the grid
- 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.
- The method `addButton` accomplishes all this and returns an object of type `tkinter.Button`

EXAMPLE

- This program (buttondemo.py) displays a single label and two command buttons.
- The buttons allow the user to clear or restore the label.
- When the user clicks Clear, the label is erased, the Clear button is disabled, and the Restore button is enabled.
- When the user clicks Restore, the label is redisplayed, the Restore button is disabled, and the Clear button is enabled.

OUTPUT



Figure 8-8 Using command buttons

PROGRAM

[illegible]

CNTD:

- To allow a program to respond to a button click, the programmer must set the button's command attribute.
- There are two ways to do this: either by supplying a keyword argument when the button is added to the window or, later, by assignment to the button's attribute dictionary.
- The value of the command attribute should be a method of no arguments, defined in the program's window class.
- The default value of this attribute is a method that does nothing

CNTD:

[illegible]

CNTD:

```
# 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"
```

Copyright 2010, Coursera, Inc. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part.

3. INPUT/OUTPUT TEXT ENTRY

- 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
- The programmer uses the method `addTextField` 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`.

CNTD:

- 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

PROGRAM TO CONVERT A STRING TO UPPERCASE

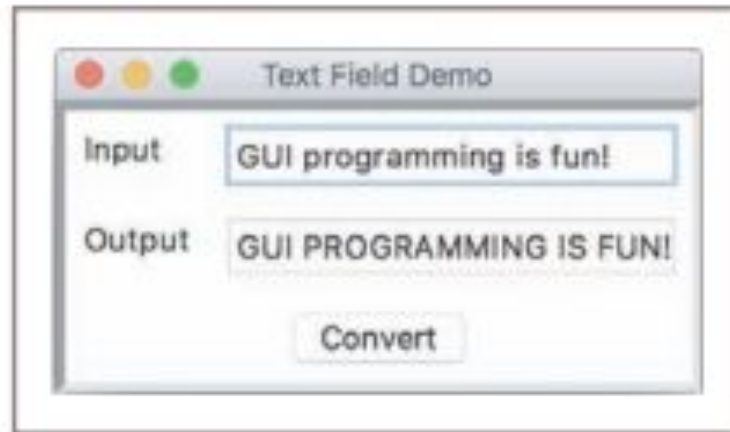


Figure 8-9 Using text fields for input and output

CNTD:

[illegible]

CNTD:

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

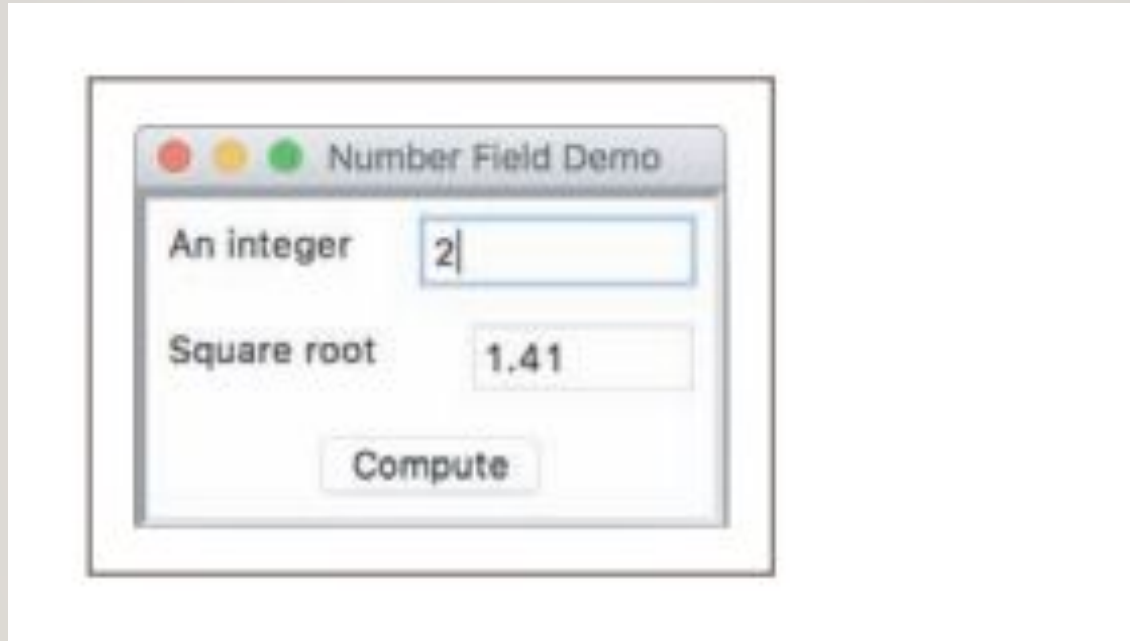
NUMERIC DATA

- breezypythongui includes two types of data fields, called IntegerField and FloatField, for the input and output of integers and floating-point numbers,
- The methods addIntegerField and addFloatField are similar in usage to the method addTextField discussed earlier.
- However, instead of an initial text attribute, the programmer supplies a value attribute.
- This value must be an integer for an integer field, but can be either an integer or a floating-point number for a float field.
- The default width of an integer field is 10 characters, whereas the default width of a float field is 20 characters.

CNTD:

- The methods `getNumber` and `setNumber` are used for the input and output of numbers with integer and float fields.
- The conversion between numbers and strings is performed automatically

COMPUTE SQUARE ROOT OF A NUMBER



CNTD:

```
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")
```


CNTD:

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


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
- Python raises an exception or runtime error when these events occur.
- The square root program raises an exception of type `ValueError` if the input datum is not an integer or is a negative integer.

TRY-EXCEPT

- In the try clause, our program attempts to input the data, compute the result, and output the result, as before.
- If an exception is raised anywhere in this process, control shifts immediately to the except clause.

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

EXAMPLE

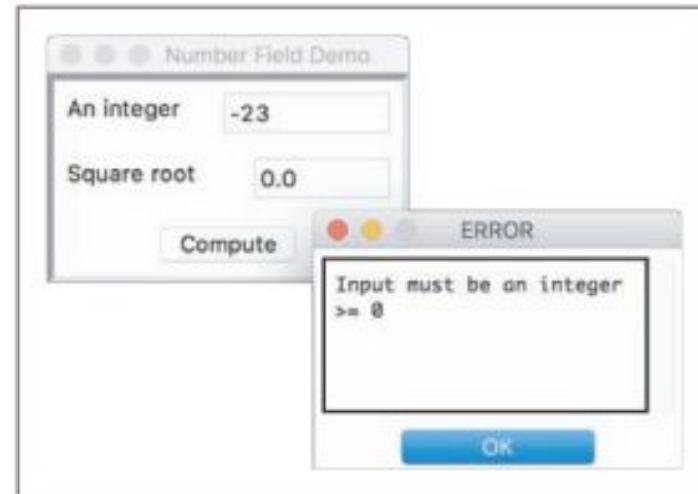


Figure 8-11 Responding to an input error with a message box

CNTD:

```
# The event handling method for the button
def computeSqrt(self):
    """Inputs the integer, computes the square root,
    and outputs the result. Handles input errors
    by displaying a message box."""
    try:
        number = self.inputField.getNumber()
        result = math.sqrt(number)
        self.outputField.setNumber(result)
    except ValueError:
        self.messageBox(title = "ERROR",
                        message = "Input must be an integer >= 0")
```


EXERCISES

1. In contrast to a terminal-based program, a GUI-based program
 - a. completely controls the order in which the user enters inputs
 - b. can allow the user to enter inputs in any order
2. The main window class in a GUI-based program is a subclass of
 - a. **TextArea**
 - b. **EasyFrame**
 - c. **Window**

3. The attribute used to attach an event-handling method to a button is named
 - a. **pressevent**
 - b. **onClick**
 - c. **command**
4. GUIs represent color values using
 - a. RGB triples of integers
 - b. hex strings
5. Multi-line text is displayed in a
 - a. text field
 - b. text area
 - c. label
6. The window component that allows a user to move the text visible beneath a **TextArea** widget is a
 - a. list box
 - b. label
 - c. scroll bar
7. The **sticky** attribute
 - a. controls the alignment of a window component in its grid cell
 - b. makes it difficult for a window component to be moved
8. A window component that supports selecting one option only is the
 - a. check button
 - b. radio button
9. A rectangular subarea with its own grid for organizing widgets is a
 - a. canvas
 - b. panel
10. The rows and columns in a grid layout are numbered starting from
 - a. (0, 0)
 - b. (1, 1)