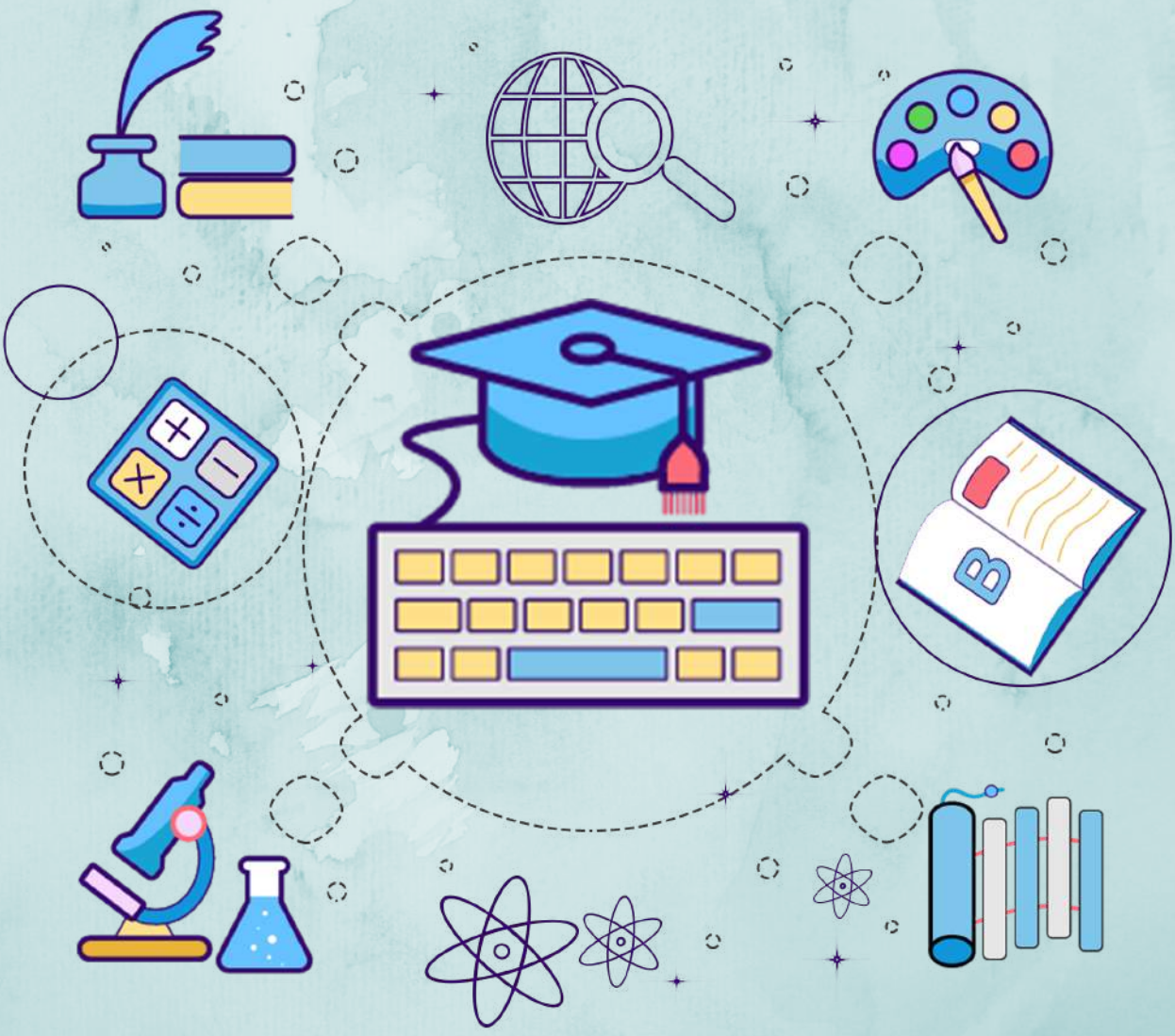
APJ ABDUL KALAM TECHNOLOGICAL UNIVERSITY

KeralaNotes



SYLLABUS | STUDY MATERIALS | TEXTBOOK

PDF | SOLVED QUESTION PAPERS

www.keralanotes.com

KeralaNotes

KTU STUDY MATERIALS

PROGRAMMING IN PYTHON

CST 362

Module 3

Related Link :

- KTU S6 CSE NOTES 2019 SCHEME
- KTU S6 SYLLABUS CSE COMPUTER SCIENCE
- KTU PREVIOUS QUESTION BANK S6 CSE SOLVED
- KTU CSE TEXTBOOKS S6 B.TECH PDF DOWNLOAD
- KTU S6 CSE NOTES | SYLLABUS | QBANK | TEXTBOOKS DOWNLOAD

MODULE 3

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 Paper tand his colleagues at MIT in the late 1960s.
- 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.**

The methods (or operations) that apply to objects of that class.

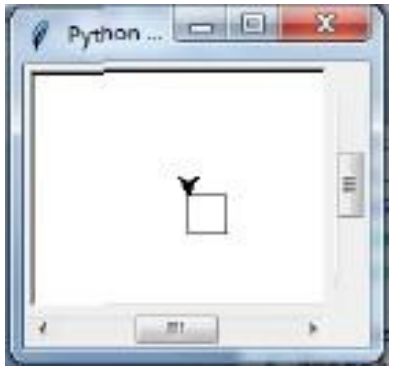
- Turtle is an object, its operations are also defined as methods.

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.

Draw a square

```
import turtle
t=turtle.Turtle()
for i in range(4):
    t.forward(50)
    t.right(90)
t.hideturtle()
```



- 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.
- The Turtle class is defined in the turtle module.
- 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
```

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)
```

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
- Pen color method is used to change the color .
- **RGB system is the common color representation system**

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

Filling Radial Patterns with Random Colors

- The Turtle class includes the pen color and fill colormethods 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")
```

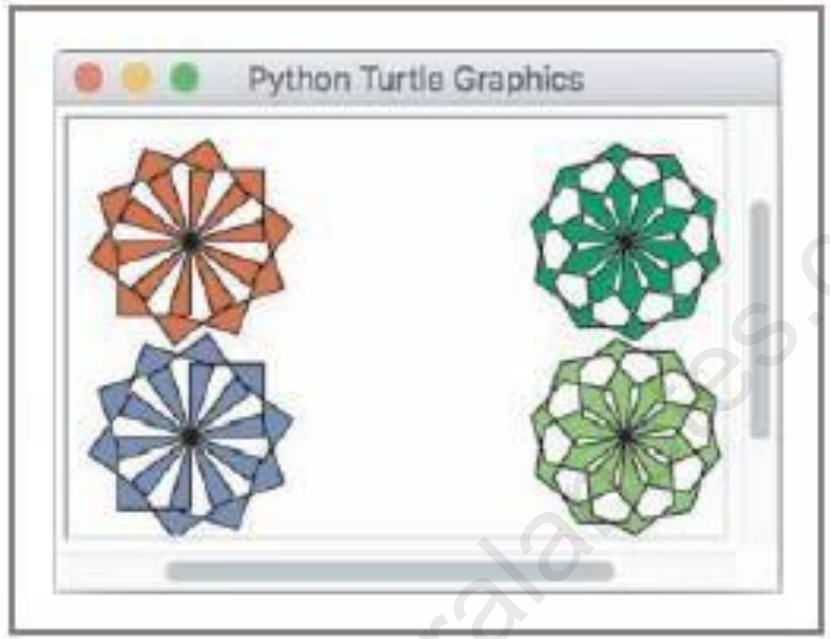


Image Processing

- An image is a visual representation of something
- An image is defined as a two dimensional function F(x,y), where x and y are spacial coordinates and amplitude of F at any pair of coordinate F(x,y) is called the intensity of that image at at that point
- When x,y and F are finite it is called as a **digital image**
- Image processing is a method to perform some operations on an image in order to get an enhanced image or to extract some useful information from it

Analog and Digital Information

- Analog Signals have continuous electrical signal whereas Digital signals have non-continuous electrical signals (bits and bytes)

Sampling and Digitizing Images

Sampling is a step used in converting an analog signal to digital signal

Or digitizing the coordinate value is called sampling

- 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/inch2= 937,500 pixels
- approximately one megapixel

Image File Formats

Jpeg

- Joint Potographic Expert Group
- Known as loosy compression-Quality of the image decrease as the file size decrease
- Animation is not supported
- Some of the image information is lost on decompression
- Support 16 million colors and mostly used in photographs

GIF

Graphic Interchange Format

- Use looseless compression algorithm
- No image information is lost on decompression
- Support only 256 colors
- Support Animations
- Mostly used in computer generated images and logos

- **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

The Properties of Images

- The software maps the bits from the image file into a rectangular area of colored 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()
```

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
```

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

- The method getPixelreturns a tupleof 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 setPixelto replace an RGB value at a given position in an image.

row-major traversal

```
>>> 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)
```

```
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:

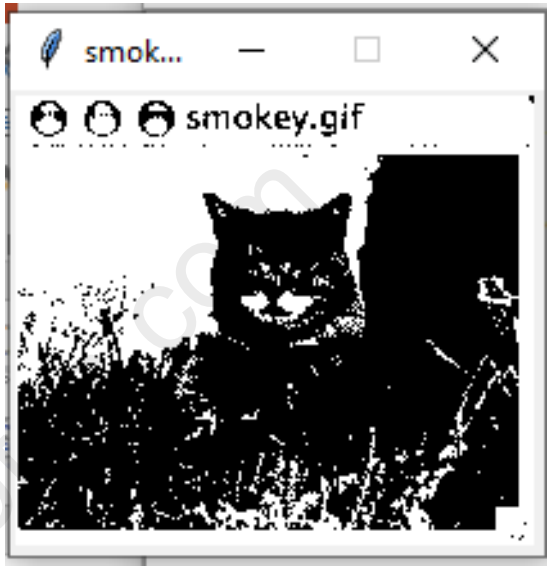
```
>>> image.setPixel(0, 0, (r + 10, g + 10, b + 10))
```

```
>>> 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

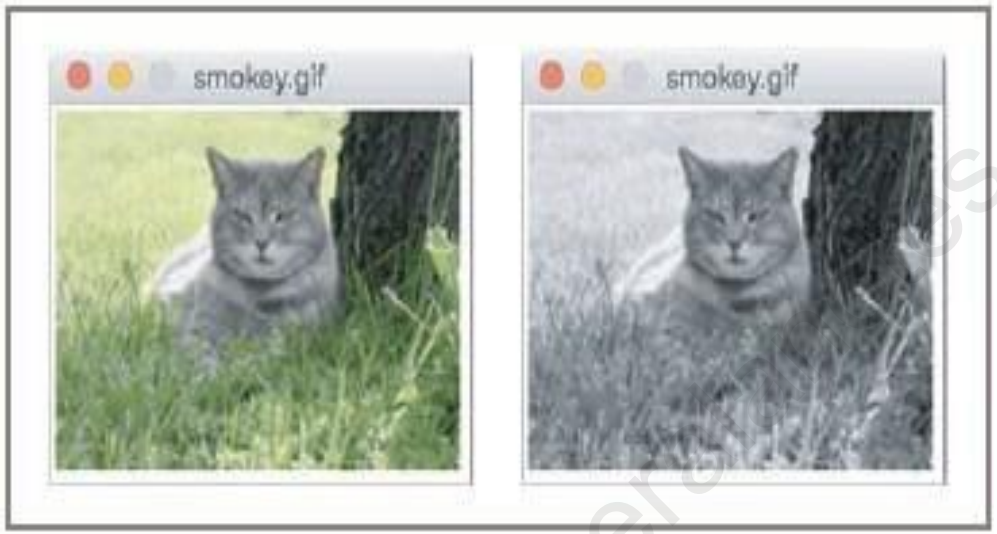
```
from images import Image
def blackAndWhite(image):
 """Converts the argument image to black and white."""
 blackPixel = (0, 0, 0)
 whitePixel = (255, 255, 255)
 for y in range(image.getHeight()):
  for x in range(image.getWidth()):
   (r, g, b) = image.getPixel(x, y)
   average = (r + g + b) // 3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

def main(filename = "smokey11.gif"):
 image = Image(filename)
 print("Close the image window to continue.")
 image.draw()
 blackAndWhite(image)
 print("Close the image window to quit.")
 image.draw()
 image.save("smokeybw.gif")
if __name__ == "__main__":
 main()
```



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.

```
from images import Image
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))

def main(filename = "smokey1.gif"):
 image = Image(filename)
 print("Close the image window to continue.")
 image.draw()
 grayscale(image)
 print("Close the image window to quit.")
 image.draw()
 image.save("smokeygs.gif")
if __name__ == "__main__":
 main()
```



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.

```
def blur(image):
 """Builds and returns a new image which is a
 blurred copy of the argument image."""
 def tripleSum(triple1, triple2):
  (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])
   #print("Oldp- ",oldP," left- ",left," right- ",right," top- ",top," bottom- ",bottom)
   #print("sums - ",sums)
   averages = tuple(map(lambda x: x // 5, sums))
   #print("averages - ", averages)
   new.setPixel(x, y, averages)
 return new
from images import Image
from functools import reduce
image = Image("smokey.gif")
image.draw()
image=blur(image)
image.draw()
```

Before blur

After blur

- **Edge Detection**
- Inverse function on a colorimage.
- 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.

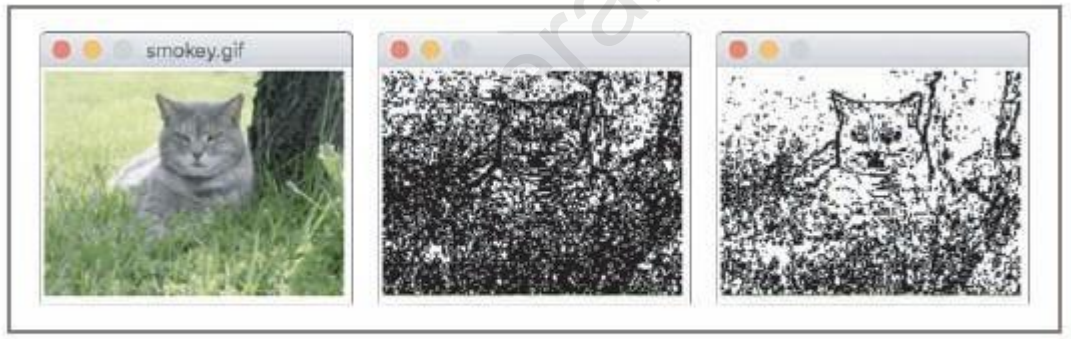


Figure 7-14 Edge detection: the original image, a luminance threshold of 10, and a luminance threshold of 20

```
from images import Image

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(r, g, b):
        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
```

```
def main(filename = "smokey11.gif"):
    image = Image(filename)
    print ("Close the image window to continue. ")
    image.draw()
    image2 = detectEdges(image, 10)
    print ("Close the image window to continue. ")
    image2.draw()
    image3 = detectEdges(image, 20)
    print ("Close the image window to continue. ")
    image3.draw()
    print ("Close the image window to quit. ")
```

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.

```
from images import Image
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
 new.draw()
 #  return new

image = Image("smokey11.gif")

shrink(image,2)
```

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

Terminal based program

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

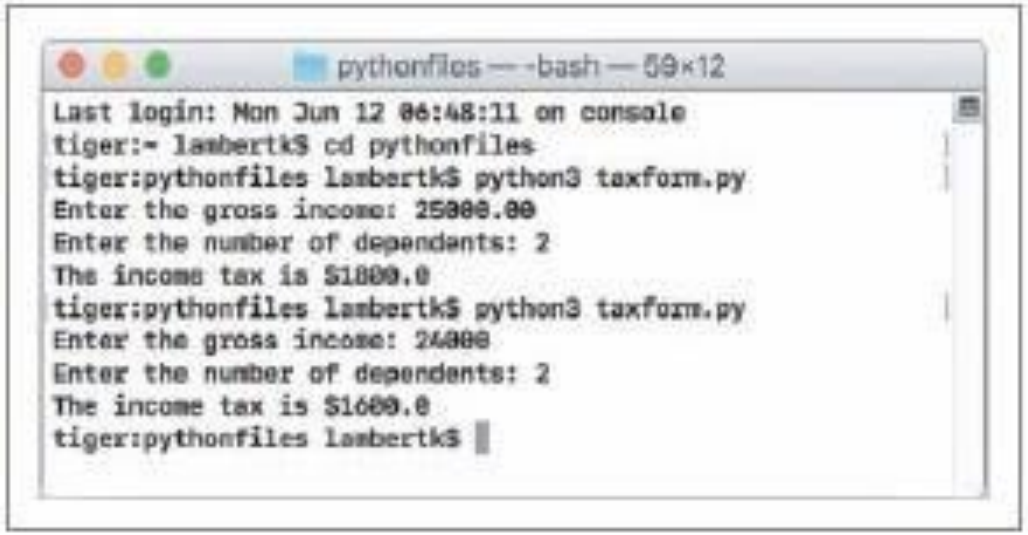
2. Prompts users to enter successive inputs

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

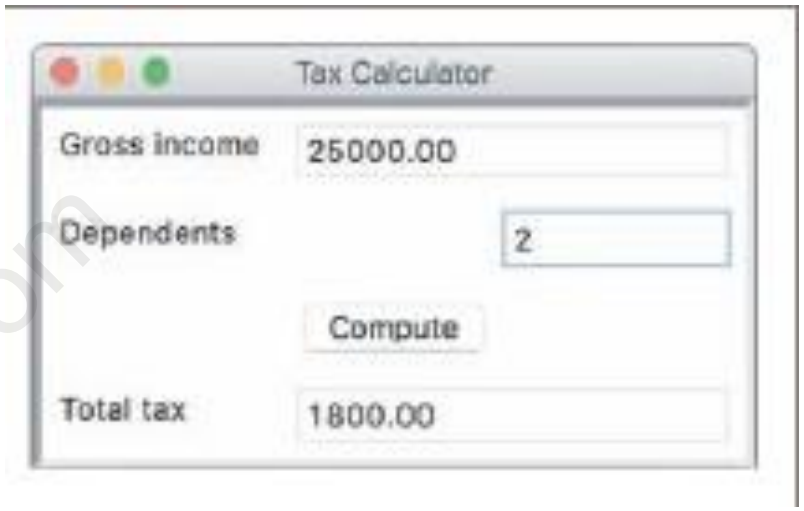
4.Once an input is entered, there is no way to back up and change it.

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.



A session with the terminal-based tax calculator program



A GUI-based tax calculator program

Window and widgets

- **Title bar**
- **Labels**
- **entry fields**
- **command button**

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

- **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**

```
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.

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

- 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.

- 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

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.

Type of Window component	Method to add it to the window
Label	addLabel(text, row, column, columnspan = 1, rowspan = 1, sticky = N+W, font = None)

Button

```
addButton(text, row, column,

          columnspan = 1, rowspan = 1,

          command = lambda: None,

          state = NORMAL)
```

FloatField(Entry)

```
addFloatField(value, row, column,
                columnspan = 1, rowspan = 1,
                width = 20, sticky = N+E,
                precision = None)
```

IntegerField(Entry)

```
addIntegerField(value, row, column,
                columnspan = 1, rowspan = 1,
                width = 10, sticky = N+E)
```

Displaying Images

One label displays the image and the other label displays the caption.

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`**).

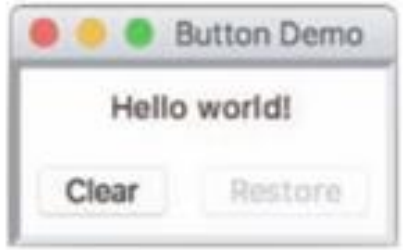
https://docs.python.org/3/library/tkinter.html#moduletkinter

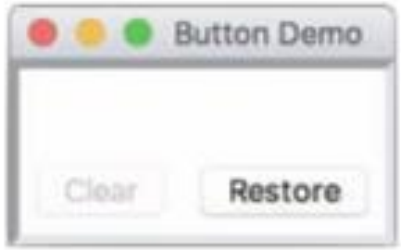
```
from breezypythongui import EasyFrame
from tkinter import PhotoImage
from tkinter.font import Font

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

Command Buttons and Responding to Events:

```
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")
```





```
# 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"
```

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.

```
def __init__(self):
    .
    .
    # Add the command button
    self.addButton(text = "Compute", row = 3, column = 0,
                   columnspan = 2, command = self.computeTax)

# The event handler method for the button
def computeTax(self):
    """Obtains the data from the input fields and uses
    them to compute the tax, which is sent to the
    output field."""
    income = self.incomeField.getNumber()
    numDependents = self.depField.getNumber()
    exemptionAmount = self.exeField.getNumber()
    tax = (income - numDependents * exemptionAmount) * .15
    self.taxField.setNumber(tax)
```

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 addTextFieldis 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 addTextFieldare text (the string to be initially displayed), row, and column.**
- **Optionalarguments 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 TextFieldmethod getTextreturns the string currently contained in a text field.
- The method setTextoutputs its string argument to a text field.

Field Type	Method	What it does
IntegerField, FloatField	**getNumber()**	Returns (as input) the number contained in the field.
	setNumber(number)	Outputs the number to the field.
FloatField	**setPrecision(width)**	Sets the number of digits to display to right of the decimal point.
TextField	**getText()**	Returns (as input) the string contained in the field.
	setText(text)	Outputs the string to the field.

Using Pop-Up Message Boxes

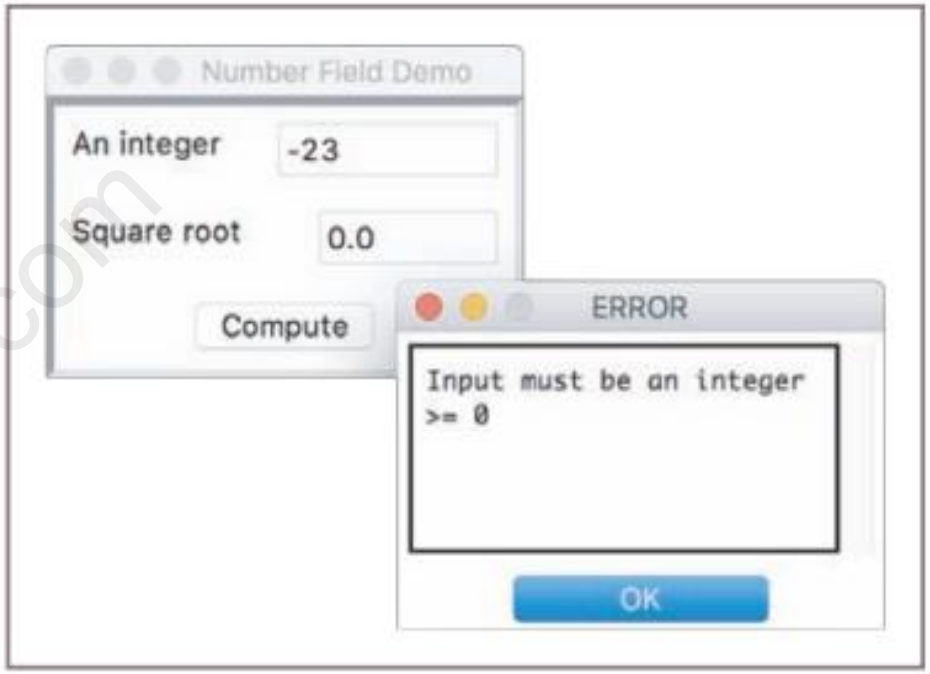
- When errors arise in a GUI-based program, the program often responds by popping up a dialog window with an error message.
- The program detects the error, pops up the dialog to inform the user, and, when the user closes the dialog, continues to accept and check input data.
- In a terminal-based program, 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.

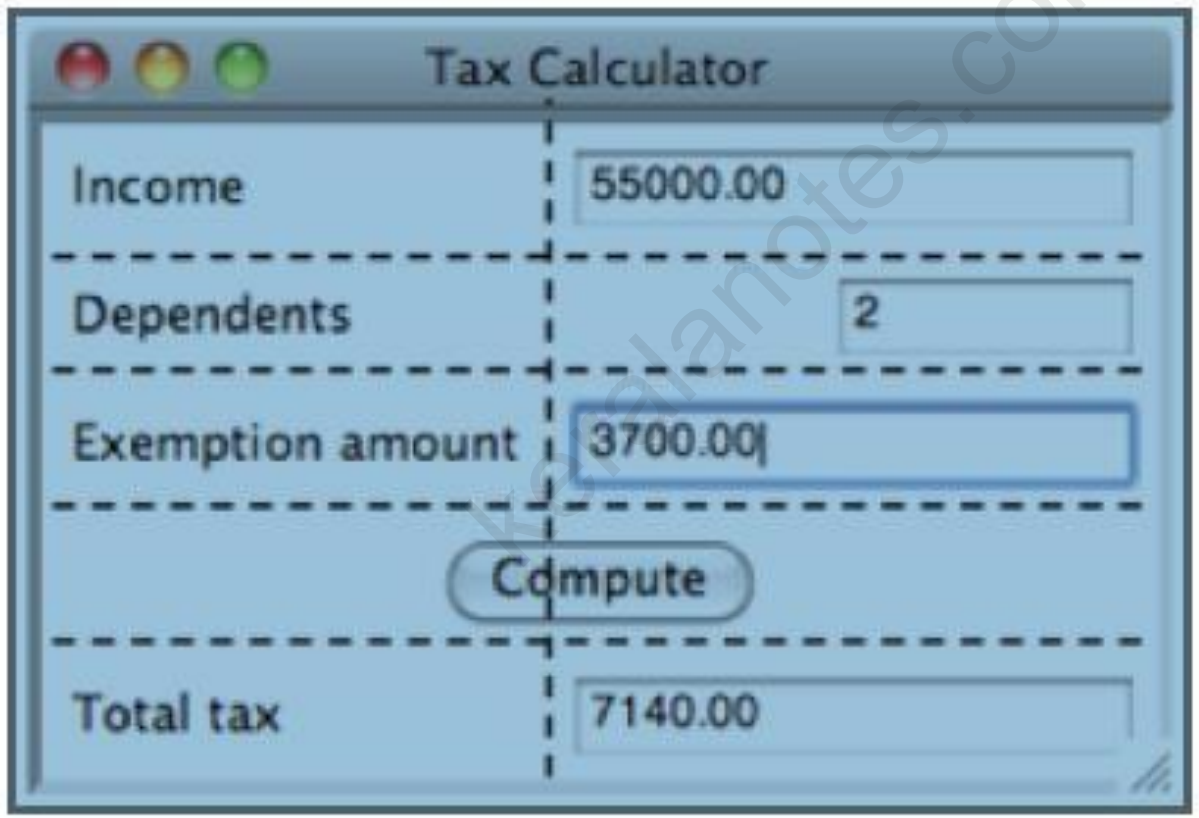
```
try:
    <statements that might raise an exception>
except <exception type>:
    <statements to recover from the exception>
```

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

```
# 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")
```



The following GUI-based application computes a person's income tax, based on a mythical tax code having a flat tax rate of 15%.



1.Layout and pop up a window with the appropriate data fields, buttons, and other controls.

2.Wait for the user to perform an action, such as pressing a button or selecting text in a field.

3. Detect a user's action (also called an event).

4.Respond appropriately to each type of user action. We call this type of programming event-driven, because the program's behavior is driven by user events

```
from breezypythongui import EasyFrame


class TaxCodeDemo(EasyFrame):

    """Application window for the tax calculator."""

    def __init__(self):
        """Sets up the window and the widgets."""
        EasyFrame.__init__(self, title = "Tax Calculator")

        # Label and field for the income
        self.addLabel(text ="Income",
                      row = 0, column = 0)
        self.incomeField = self.addFloatField(value = 0.0,
                                              row = 0,
                                              column = 1)
```

```
# Label and field for the number of dependents
self.addLabel(text = "Dependents",
              row = 1, column = 0)
self.depField = self.addIntegerField(value = 0,
                                      row = 1,
                                      column = 1)

# Label and field for the exemption amount
self.addLabel(text = "Exemption amount",
              row = 2, column = 0)
self.exeField = self.addFloatField(value = 0.0,
                                   row = 2,
                                   column = 1)
# The command button
self.addButton(text = "Compute", row = 3, column = 0,
               columnspan = 2, command = self.computeTax)
```

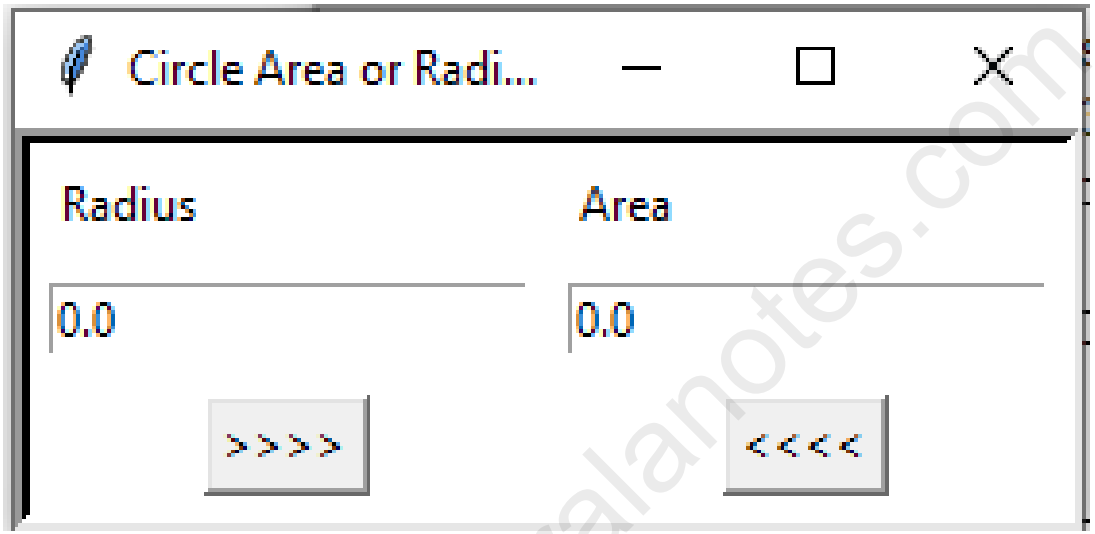
```
        # Label and field for the tax
        self.addLabel(text = "Total tax",
                      row = 4, column = 0)
        self.taxField = self.addFloatField(value = 0.0,
                                           row = 4,
                                           column = 1,
                                           precision = 2)


    # The event handler method for the button
    def computeTax(self):
        """Obtains the data from the input fields and uses
        them to compute the tax, which is sent to the
```

```
        output field."""
        income = self.incomeField.getNumber()
        numDependents = self.depField.getNumber()
        exemptionAmount = self.exeField.getNumber()
        tax = (income - numDependents * exemptionAmount) * .15
        self.taxField.setNumber(tax)

#Instantiate and pop up the window.
TaxCodeDemo().mainloop()
```

Create GUI program for computing the area or radius of a circle

```
from breezypythongui import EasyFrame
import math

class CircleArea(EasyFrame):
    """Computes the area or the radius of a circle."""

    def __init__(self):
        """Sets up the window and widgets."""
        EasyFrame.__init__(self, title = "Circle Area or Radius")

        # Label and field for the radius
        self.addLabel(text = "Radius",
                      row = 0, column = 0)
        self.radiusField = self.addFloatField(value = 0.0,
                                              row = 1,
                                              column = 0)

        # Label and field for the area
        self.addLabel(text = "Area",
                      row = 0, column = 1)
        self.areaField = self.addFloatField(value = 0.0,
                                            row = 1,
                                            column = 1)

        # Radius to area button
        self.addButton(text = ">>>>",
                       row = 2, column = 0,
                       command = self.computeArea)

        # Area to Radius button
        self.addButton(text = "<<<<",
                       row = 2, column = 1,
                       command = self.computeRadius)
```

```
    # The event handler method for the area button
    def computeArea(self):
        """Inputs the radius, computes the area,
        and outputs the area."""
        radius = self.radiusField.getNumber()
        area = radius ** 2 * math.pi
        self.areaField.setNumber(area)

    # The event handler method for the radius button
    def computeRadius(self):
        """Inputs the area, computes the radius,
        and outputs the radius."""
        area = self.areaField.getNumber()
        radius = math.sqrt(area / math.pi)
        self.radiusField.setNumber(radius)

#Instantiate and pop up the window.
CircleArea().mainloop()
```



Important Questions

- Differentiate between terminal based and GUI based programming in Python?
- Write a program to draw a hexagon using turtle
- Write a note on the image processing in Python
- Describe the features of event driven Programming?
- Write a GUI based program that allows the user to convert temperature values between degrees Fahrenheit and degrees Celsius. The interface should have labeled entry fields for these two values. These components should e arranged in a grid where the labels occupy the first row and the corresponding field occupy the second row. At start up the Fahrenheit field should contain 32 and the Celsius field contain 0.0.The third row in the window contain two command buttons ,labeled >>>> and <<<<.When the user presses the first button, the program should use the data in the Fahrenheit field to compute the Celsius value, which should then be output to the Celsius field. The second button should perform the inverse function?

Thank you