Module -3 (Graphics) (7 hours)

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.

Terminal-Based Program

- ➔ The terminal-based program prompts the user for user inputs and other program dependant values.
- ➔ After the user enters his inputs, the program responds by computing and displaying the results.
- ➔ The program then terminates execution.

terminal-based user interface

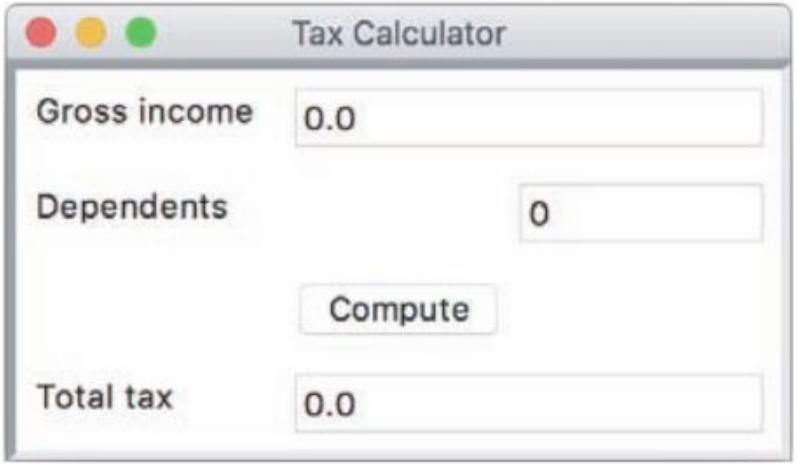
- ➔ This terminal-based user interface has several obvious effects on its users:
  - ◆ The user is constrained to reply to a definite sequence of prompts for inputs.
  - ◆ Once an input is entered, there is no way to back up and change it.
  - ◆ To obtain results for a different set of input data, the user must run the program again and all of the inputs must be re-entered.
- ➔ These problems for users can be solved by converting the interface to a GUI .

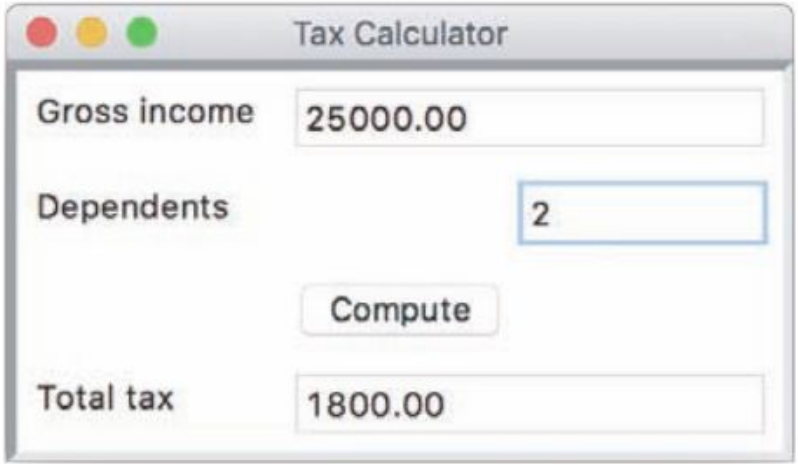
Program computes and displays a person's income tax [terminal based]

```
Enter the gross income: 25000.00
Enter the number of dependents: 2
The income tax is $1800.0
```

VSS

GUI-Based Version





VSS

The GUI-Based Version

- The GUI-based version of the program displays a window that contains various components, also called widgets.
- A GUI program is event driven, that it is inactive until the user clicks a button or selects a menu option.
- A title bar at the top of the window.
  - This bar contains the title of the program, “Tax Calculator.”
  - three colored disks. Each disk is a command button .
  - The user can use the mouse to click the left disk to quit the program
  - the middle disk to minimize the window, or the right disk to zoom the window.
  - The user can also move the window around the screen by holding the left mouse button on the title bar and dragging the mouse.

The GUI-Based Version

- A set of labels along the left side of the window.
  - These are text elements that describe the inputs and outputs. For example, “Gross income” is one label.
- A set of entry fields along the right side of the window.
  - These are boxes within which the program can output text or receive it as input from the user.
- A single command button labeled Compute.
  - When the user uses the mouse to press this button, the program responds by using the data in the two input fields to compute the income tax.
- The user can also alter the size of the window by holding the mouse on its lower-right corner and dragging in any direction.

The GUI-Based Version

effects on users:

- ➔ The user is not constrained to enter inputs in a particular order. Before she presses 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.

Event-Driven Programming

- ➔ 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 .

Assignment:

Describe the differences between terminal-based user interfaces and GUIs.

Graphics

- Graphics is the discipline that underlies the representation and display of geometric shapes in two- and three-dimensional space, as well as image processing.
-

Simple Graphics using Turtle

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

VSS

turtle attributes

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.

VSS

Turtle Operations

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.

VSS

Turtle Operations

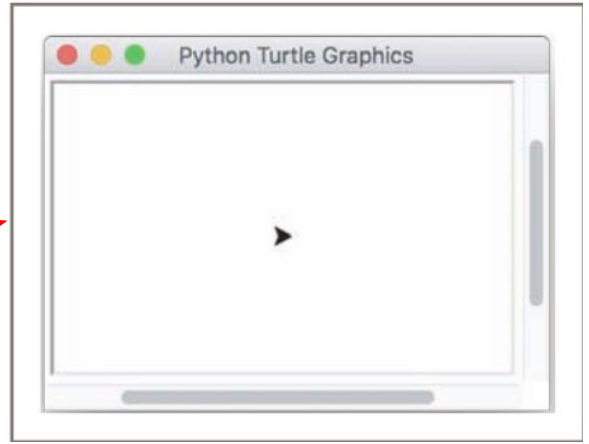
`t.pencolor(string)`	to the specified string, such as `"red"`. Returns the curr color of `t` when the arguments are omitted.
`t.fillcolor(r, g, b)` `t.fillcolor(string)`	Changes the fill color of `t` to the specified RGB value or the specified string, such as `"red"`. Returns the current color of `t`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

VSS

Object Instantiation and the turtle Module

- Before use a Turtle object, must create them.
- That is create an instance of the object's class.
- The process of creating an object is called
- instantiation .

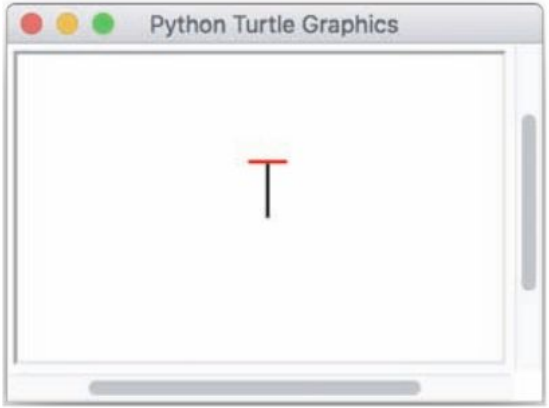
```
from turtle import Turtle

t = Turtle()
```



- A window is created with 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.

Drawing vertical and horizontal lines for the letter T

```
>>> 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
```



VSS

Drawing Two-Dimensional Shapes

```
from turtle import Turtle
import time
t = Turtle()
def square(t, length):
 for count in range(4):
  t.forward(length)
  t.left(90)
square(t, 30)
```

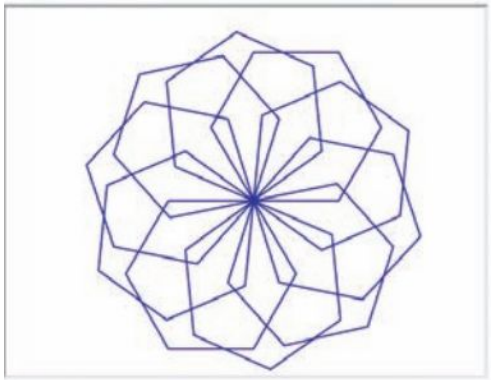
VSS

function to draw a hexagon:

```
def hexagon(t, length):
        for count in range(6):
                t.forward(length)
                t.left(60)
from turtle import Turtle
import time
t = Turtle()
hexagon(t,10)
```

radialHexagons [A radial pattern with 10 hexagons]

```
def hexagon(t, length):
        for count in range(6):
                t.forward(length)
                t.left(60)
def radialHexagons(t, n, length):
        for count in range(n):
                hexagon(t,length)
                t.left(360 / n)
from turtle import Turtle
import time
t = Turtle()
radialHexagons(t,10,50)
```



VSS

squares and hexagons [general function]:

polygons.py

```
def square(t, length):
        for count in range(4):
                t.forward(length)
                t.left(90)
def hexagon(t, length):
        for count in range(6):
                t.forward(length)
                t.left(60)
```

```
def radialpattern(t, n, length, shape):
        for count in range(n):
                shape(t, length)
                t.left(360 / n)
from turtle import Turtle
from polygons import *
import time
t = Turtle()
radialpattern(t, n = 10, length = 50, shape = square)
t.clear()
radialpattern(t,10,50,hexagon)
```

VSS

Examining an Object's Attributes

- The Turtle methods which modify a Turtle object's attributes, such as its position, heading, and color etc are called mutator methods .
  - they change the internal state of a Turtle object.
- The Turtle methods like position() , simply return the values of a Turtle object's attributes without altering its state. These methods are called accessor methods .

Manipulating a Turtle's Screen

- ➔ Turtle object is associated with instances of the classes Screen and Canvas.
- ➔ The Screen object's attributes include its width and height in pixels, and its background color, among other things.
- ➔ access a turtle's Screen object using the notation t.screen
- ➔ methods window_width() and w indow_height() can be used to locate the boundaries of a turtle's window.

Colors and the RGB System

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

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)

Turtle Fill

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

We can use **.fillcolor(), .begin_fill()** and **.end_fill()** methods.

A typical fill operation in turtle can be like this:

- Define a filling color using **turtle.fillcolor()**
- Start the filling operation with this command: **turtle.begin_fill()**
- Perform your turtle drawing in the middle
- End the filling operation with this command: **turtle.end_fill()**

Turtle Fill

.fillcolor()

Filling a drawing in turtle is pretty simple.

You just need to place your drawing between **begin_fill()** and **end_fill()** commands.

A typical fill operation in turtle can be like this:

- Define a filling color using **turtle.fillcolor()**
- End the filling operation with this command: **turtle.end_fill()**

Turtle Fill

.begin_fill()

This method activates filling operation in turtle. It should be placed before the drawing of a pattern.

It also makes sense to include a **turtle.fillcolor()** object before the **turtle.begin_fill()** object.

- Define a filling color using **turtle.fillcolor()**
- End the filling operation with this command: **turtle.end_fill()**

Turtle Fill

.end_fill()

Filling a drawing in turtle is pretty simple.

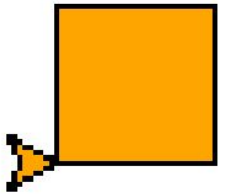
You just need to place your drawing between **begin_fill()** and **end_fill()** commands.

A typical fill operation in turtle can be like this:

- Define a filling color using **turtle.fillcolor()**
- End the filling operation with this command: **turtle.end_fill()**

Example: Fill Sqaure

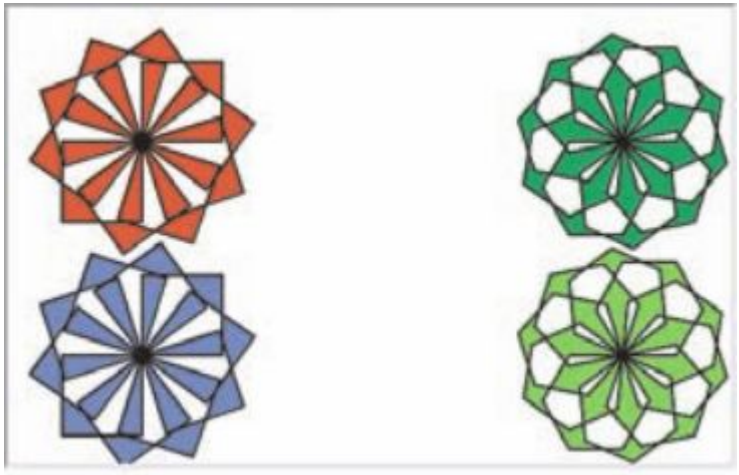
```
import turtle
t = turtle.Turtle()
def square(t, length):
    t.fillcolor("orange")
    t.begin_fill()
    #"""Draws a square with the given length."""
    for count in range(4):
        t.forward(length)
        t.left(90)
    t.end_fill()
square(t,30)
```



VSS

Case study:

Draws a radial pattern of squares in a random fill color at each corner of the window.



Case Study: Recursive Patterns in Fractals

Write a program that allows the user to draw a particular c-curve at varying levels.

Image Processing

Theory: Refer chapter 7 : Topics-

- ➔ Image Processing
  - Analog and Digital Information
  - Sampling and Digitizing Images
  - Image File Formats
  - Image-Manipulation Operations
  - The Properties of Images
  - The images Module

The images Module

- This package of resources, named images , 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.

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.

The images Module - Display an image:

Imports the Image class from the images module

2. Instantiates this class using the file named smokey.gif

3. Draws the image

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

VSS

examine its width and height, as follows:

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

print the image's string representation:

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

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

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

setPixel to replace an RGB value at a given position

```
>>> 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()
```

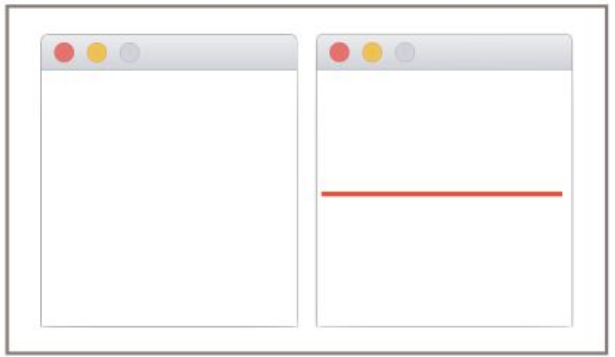


Figure 7-10 An image before and after replacing the pixels

The save operation to write an image back to an existing file using the current filename.

saves the new image using the filename horizontal.gif

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

A Loop Pattern for Traversing a Grid:

- ➔ uses a nested loop structure to traverse a two-dimensional grid of pixels.
- ➔ Each data value in the grid is accessed with a pair of coordinates using the form (<column>, <row>) .

```
>>> 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)
```

```
for y in range(height):
    for x in range(width):
        <do something at position (x, y)>
```

VSS

fill red in blank image:

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

VSS

A Word on Tuples:

- ➔ a pixel's RGB values are stored in a tuple
- ➔ Eg: increase each of a pixel's RGB values by 10, thereby making the pixel brighter.

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

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

Converting an Image to Black and White

```
from images import Image
image = Image("flower.gif")
image.draw()
def blackAndWhite(image):
        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)
blackAndWhite(image)
image.draw()
```

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 .

```
def grayscale(image):
    """Converts the argument image to graysca
    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))
```

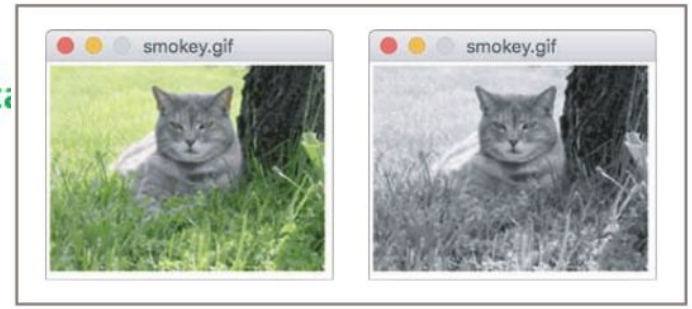


Figure 7-13 Converting a color image to grayscale

VSS

Copying an Image

- The Image class includes a clone method , 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.

```
>>> from images import Image
>>> image = Image("smokey.gif")
>>> image.draw()
>>> newImage = image.clone()    # Create a copy of image
>>> newImage.draw()
```

Blurring an Image

- ➔ This algorithm resets each pixel's color to the average of the colors of the four pixels that surround it.
- ➔ The function blur expects an image as an argument and returns a copy of that image with blurring.
- ➔ The function blur begins its traversal of the grid with position (1, 1) and ends with position (width 2 2, height 2 2).
- ➔ means that the algorithm does not transform the pixels on the image's outer edges.

VSS

```
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
```

This function expects two tuples of integers as arguments and returns a single tuple containing the sums of the values at each position.

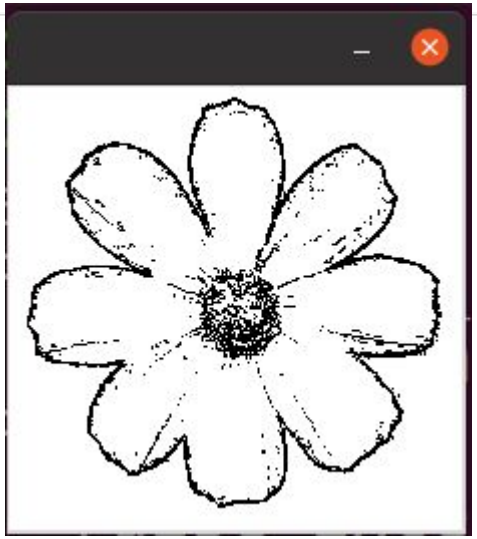
five tuples of RGB values are wrapped in a list and passed with the tripleSum function to the reduce function. This function repeatedly applies tripleSum to compute the sums of the tuples, until a single tuple containing the sums is returned.

lambda function is mapped onto the tuple of sums, and the result is converted to a tuple. The lambda function divides each sum by 5. Thus, you are left with a tuple of the average RGB values.

VSS

Edge Detection

```
from images import Image
image = Image("flower.gif")
amount=20
def detectEdges(image, amount):
 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
image2=detectEdges(image, amount)
image2.draw()
```



Reducing the Image Size

- The size and the quality of an image on a display medium, such as a computer monitor or a printed page, depend on two factors:
  - the image's width and height in pixels a
  - The display medium's resolution .
  - Resolution is measured in pixels, or dots per inch (DPI).
- When the resolution of a monitor is increased, the images appear smaller, but their quality increases.
- Conversely, when the resolution is decreased, images become larger, but their quality degrades.

```
from images import Image
image = Image("flower.gif")
def shrink(image, factor):
 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
 print(newX)
 print(newY)
 return new
image1=shrink(image, 2)
```

Sample Questions:

1. Explain the advantages and disadvantages of lossless and lossy image file-compression schemes.
2. The size of an image is 1680 pixels by 1050 pixels. Assume that this image has been sampled using the RGB color system and placed into a raw image file. What is the minimum size of this file in megabytes? (*Hint*: There are 8 bits in a byte, 1024 bits in a kilobyte, and 1000 kilobytes in a megabyte.)
3. Describe the difference between Cartesian coordinates and screen coordinates.
4. Describe how a row-major traversal visits every position in a two-dimensional grid.
5. How would a column-major traversal of a grid work? Write a code segment that prints the positions visited by a column-major traversal of a 2-by-3 grid.
6. Explain why one would use the **`clone`** method with a given object.
7. Why does the **`blur`** function need to work with a copy of the original image?

VSS

Coding Simple GUI-Based Programs

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

What Is breezypythongui?

- ➔ A module of classes and functions that makes GUI programming with tkinter easy for beginners

First GUI Program: Hello World

Import the EasyFrame class from the breezypythongui module. This class is a sub class of tkinter 's Frame class, which represents a top-level window.

```
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.
if __name__ == "__main__":
    LabelDemo().mainloop()
```

Define the LabelDemo class as a subclass of EasyFrame . The LabelDemo class describes the window's layout and functionality for this application.

VSS

First GUI Program: Hello World

- 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).

First GUI Program: Hello World

- ➔ 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. 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
- ➔ 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.
if __name__ == "__main__":
    <class name>().mainloop()
```

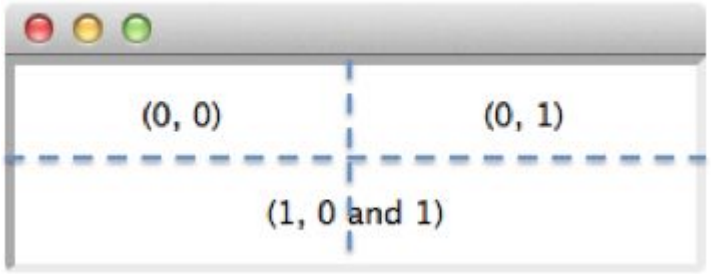
VSS

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 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.

Alignment and Spanning

```
def __init__(self):
    """Sets up the window and the labels."""
    EasyFrame.__init__(self)
    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)
```



VSS

Windows and Window Components : Window Attributes

- ➔ width (window shrink-wraps around widgets by default)
- ➔ height
- ➔ title (empty string by default)
- ➔ background (“white” by default)
- ➔ resizable (True by default)

Can be modify using

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

- Another way to change a window’s attributes is to reset them in the window’s attribute dictionary .
- Each window or window component maintains a dictionary of its attributes and their values.
- To access or modify an attribute, can use the standard subscript notation with the attribute name as a dictionary key.

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

- self in this case refers to the window itself.

To change a window's attributes EasyFrame class includes the four methods

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`**.

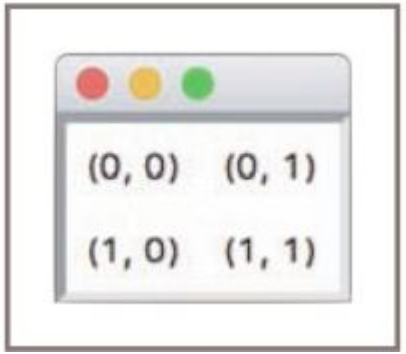
Optional Arguments to __init__

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

VSS

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.




```
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)
```

vSS

Window Layout

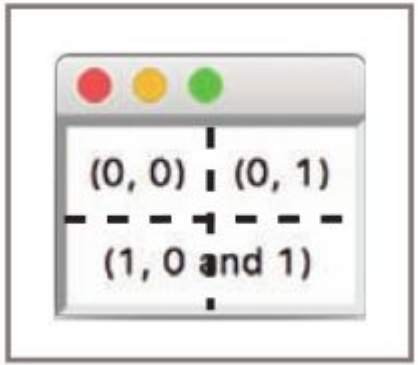
- ➔ Each type of window component has a default alignment within its grid position.
- ➔ default alignment is northwest.
- ➔ The programmer can override the default alignment by including the sticky attribute
- ➔ The values of sticky are the strings “N,” “S,” “E,” and “W,” or any combination thereof.

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

Window Layout :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

```
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)
```



VSS

Types of Window Components and Their Attributes

- GUI programs uses several types of window components,
- These include labels, entry fields, text areas, command buttons, drop-down menus, sliding scales, scrolling list boxes, canvases, and many others.
- self.addComponentType(<arguments>)
- When this method is called, breeypythongui
  - 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.

VSS

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.

VSS

Code for 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"
```

- This program adds two labels to the window. One label displays the image and the other label displays the caption.
- The image label is first added to the window with an empty text string.
- The program then creates a PhotoImage object from an image file and sets the image attribute of the image label to this object.
- the variable used to hold the reference to the image must be an instance variable (prefixed by self)

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`**).

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, with event
        # handler methods supplied.
        self.clearBtn = self.addButton(text = "Clear",
                                       row = 1, column = 0,
                                       command = self.clear)
        self.restoreBtn = self.addButton(text = "Restore",
                                         row = 1, column = 1,
                                         state = "disabled",
                                         command = self.restore)
```

```
# 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"
```

VSS

Input and Output with Entry Fields

Text Fields

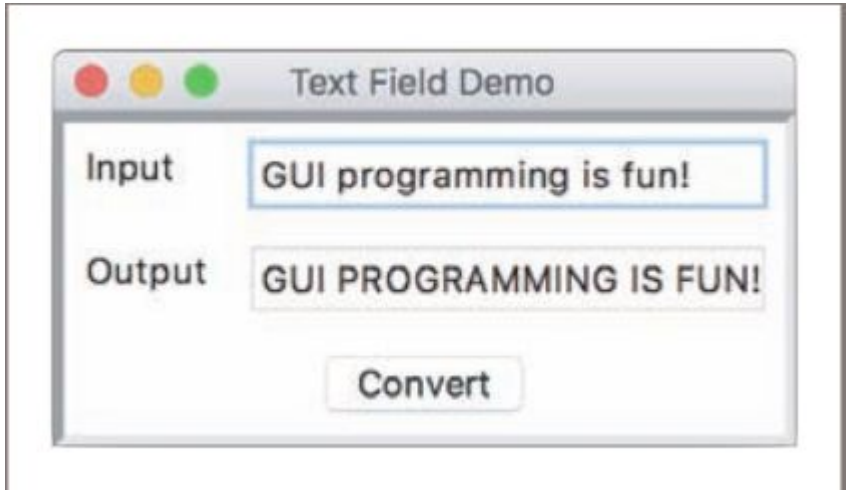
- for entering or displaying a single-line string of characters.
- method addTextField adds a text field to a window.
- The method returns an object of type TextField , which is a subclass of tkinter.Entry .
- Required arguments to addTextField are text (the string to be initially displayed), row , and column .
- Optional arguments are rowspan , columnspan , sticky , width , and state .

- A text field is aligned by default to the northeast of its grid cell.
- A text field has a default width of 20 characters.
- TextField method getText returns the string currently contained in a text field.
-

```
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)
```

/SS

```
# 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)
```



VSS

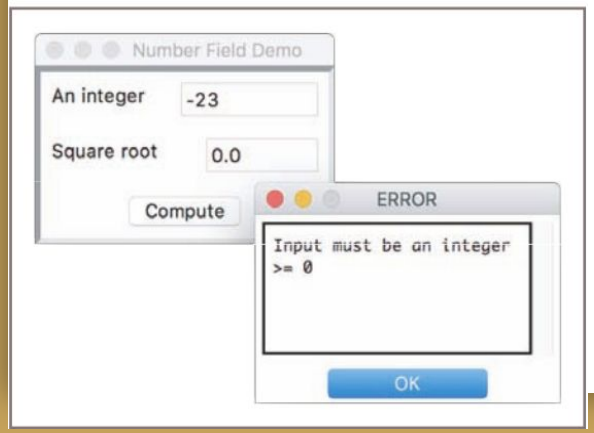
Using Pop-Up Message Boxes

- ➔ When errors arise in a GUI-based program, the program 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.
- ➔ Python raises an exception or runtime error when these events occur.

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

VSS

```
# 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")
```



VSS

Check Buttons

➔ check button consists of a label and a box that a user can select or deselect with the mouse.

```
self.chickCB = self.addCheckbutton(text = "Chicken",
                                   row = 0, column = 0)

self.taterCB = self.addCheckbutton(text = "French fries",
                                   row = 0, column = 1)

self.beanCB = self.addCheckbutton(text = "Green beans",
                                  row = 1, column = 0)

self.sauceCB = self.addCheckbutton(text = "Applesauce",
                                   row = 1, column = 1)
```

```
if self.chickCB.isChecked():
```

VSS

Radio Buttons

- ➔ Check buttons allow a user to select multiple options in any combination.
- ➔ When the user must be restricted to one selection only, the set of options can be presented as a group of radio buttons .

Case Study: The Guessing Game

Refer Textbook.