Python Programming Essentials

Unit 1: Basic Python

CHAPTER 4: OBJECTS AND GRAPHS

DR. ERIC CHOU

IEEE SENIOR MEMBER

Objectives

- To understand the concept of objects and how they can be used to simplify programs.
- To be familiar with the various objects available in the graphics library.
- To be able to create objects in programs and call appropriate methods to perform graphical computations.

Objectives

- To understand the fundamental concepts of computer graphics, especially the role of coordinate systems and coordinate transformations.
- To understand how to work with both mouse and text-based input in a graphical programming context.
- To be able to write simple interactive graphics programs using the graphics library.

Overview

LECTURE 1

Overview

- Each data type can represent a certain set of values, and each had a set of associated operations.
- The traditional programming view is that data is passive – it's manipulated and combined with active operations.

Overview

- Modern computer programs are built using an **object-oriented** approach.
- Most applications you're familiar with have Graphical User Interfaces (GUI) that provide windows, icons, buttons and menus.
- There's a graphics library (**graphics.py**) written specifically to go with this book. It's based on **Tkinter**.

The Object of Objects

LECTURE 2

The Object of Objects

- Basic idea – view a complex system as the interaction of simpler objects. An **object** is a sort of active data type that combines data and operations.
- Objects **know stuff** (contain data) and they can **do stuff** (have operations).
- Objects interact by sending each other **messages**.

The Object of Objects

- Suppose we want to develop a data processing system for a college or university.
- We must keep records on students who attend the school. Each student will be represented as an object.

The Object of Objects

- The student object would contain data like:
  - Name
  - ID number
  - Courses taken
  - Campus Address
  - Home Address
  - GPA
  - Etc.

The Object of Objects

- The student object should also respond to requests.
- We may want to send out a campus-wide mailing, so we'd need a campus address for each student.
- We could send the **`printCampusAddress`** to each student object. When the student object receives the message, it prints its own address.

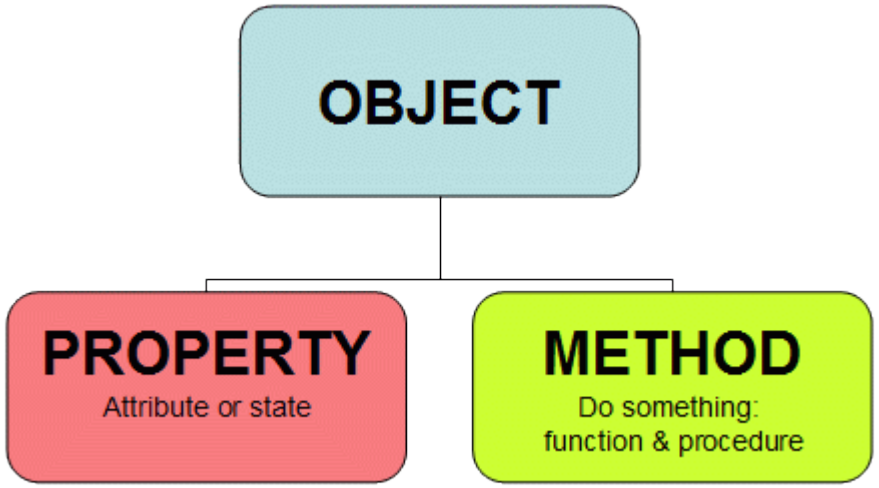
Object of Objects

- Objects may refer to other objects.
- Each course might be represented by an object:
  - Instructor
  - Student roster
  - Prerequisite courses
  - When and where the class meets

Object of Objects

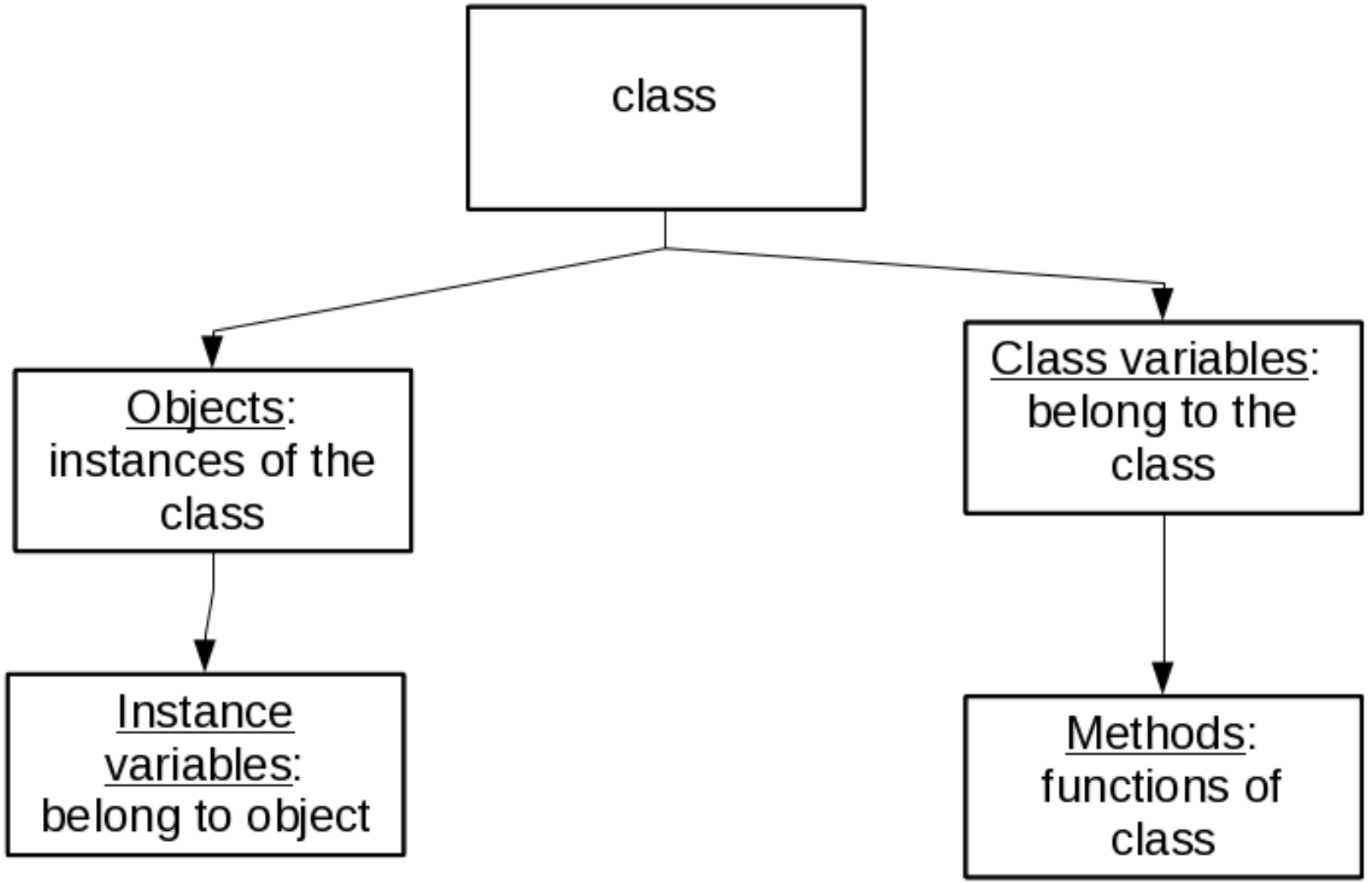
- Sample Operation
  - `addStudent`
  - `delStudent`
  - `changeRoom`
  - `Etc.`

Class VS Objects



Class is Object Type Declaration
Object is Instance is the real data/method entity in a program.

Object Oriented Programming



Simple Graphics Programming

LECTURE 3

Simple Graphics Programming

- This chapter uses the **graphics.py** library supplied with the supplemental materials.
- Two location choices
  - In Python's Lib directory with other libraries
  - In the same folder as your graphics program

Simple Graphics Programming

- Since this is a library, we need to import the graphics commands

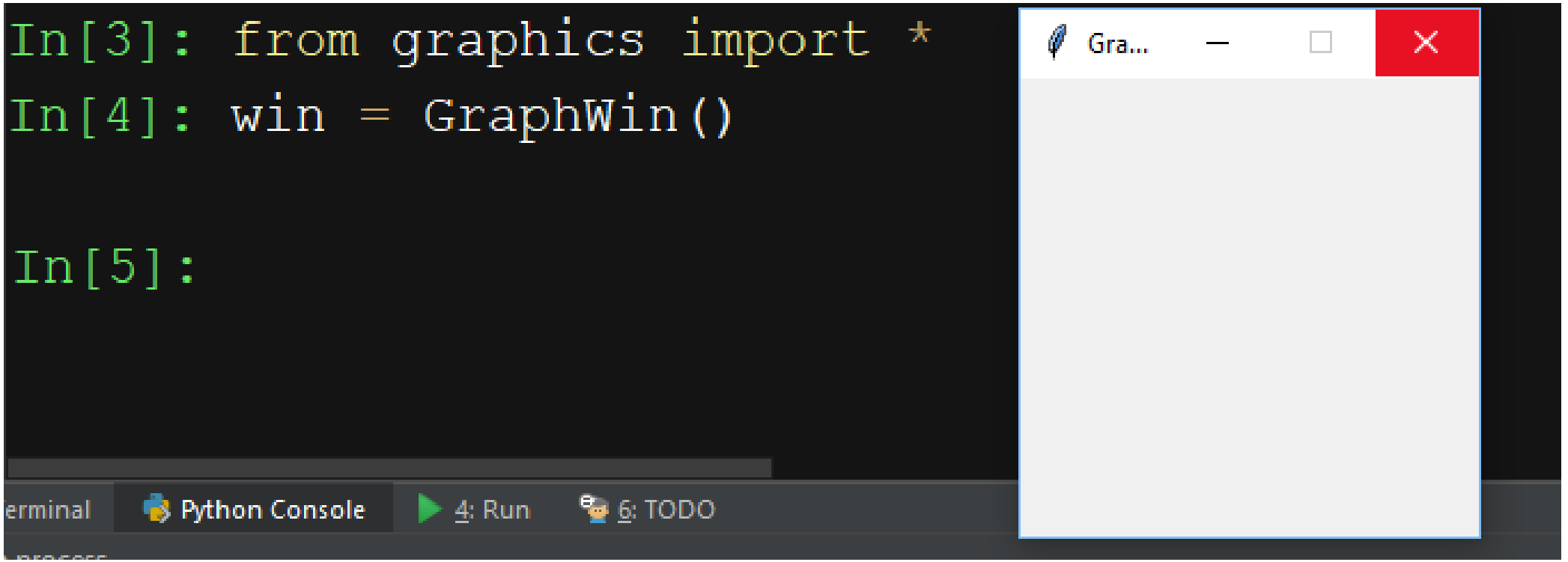
```
>>> import graphics
```

- A graphics window is a place on the screen where the graphics will appear.

```
>>> win = graphics.GraphWin()
```

- This command creates a new window titled “Graphics Window.”

Simple Graphics Programming



Simple Graphics Programming

- **GraphWin** is an object assigned to the variable **win**. We can manipulate the window object through this variable, similar to manipulating files through file variables.
- Windows can be closed/destroyed by issuing the command

```
>>> win.close()
```

Simple Graphics Programming

- It's tedious to use the `graphics.` notation to access the graphics library routines.
- `from graphics import *`

  The "from" statement allows you to load specific functions from a library module. "*" will load all the functions, or you can list specific ones.

Simple Graphics Programming

- Doing the import this way eliminates the need to preface graphics commands with graphics.

```
>>> from graphics import *
>>> win = GraphWin()
```

Simple Graphics Programming

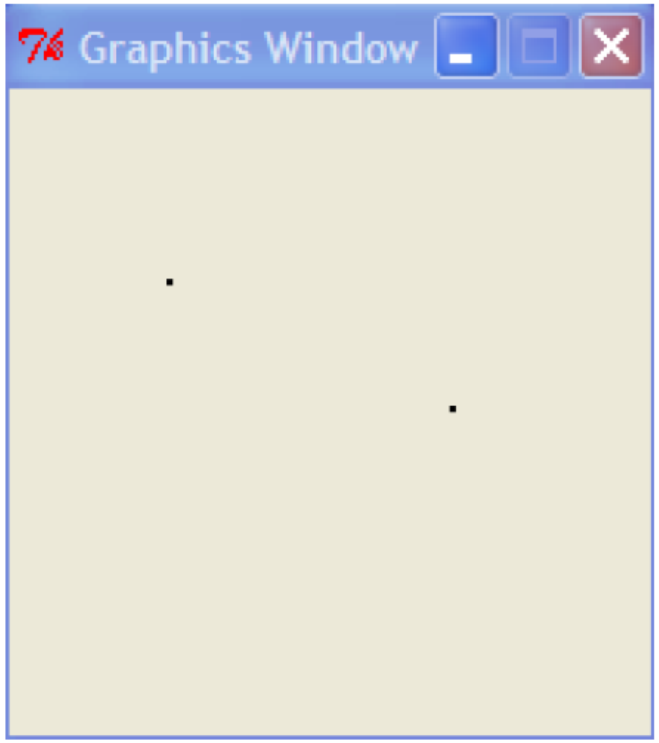
- A graphics window is a collection of points called **pixels** (picture elements).
- The default GraphWin is 200 pixels tall by 200 pixels wide (40,000 pixels total).
- One way to get pictures into the window is one pixel at a time, which would be tedious. The graphics library has a number of predefined routines to draw geometric shapes.

Simple Graphics Programming

- The simplest object is the **`Point`**. Like points in geometry, point locations are represented with a coordinate system (x, y), where x is the horizontal location of the point and y is the vertical location.
- The origin (0,0) in a graphics window is the upper left corner.
- X values increase from left to right, y values from top to bottom.
- Lower right corner is (199, 199)

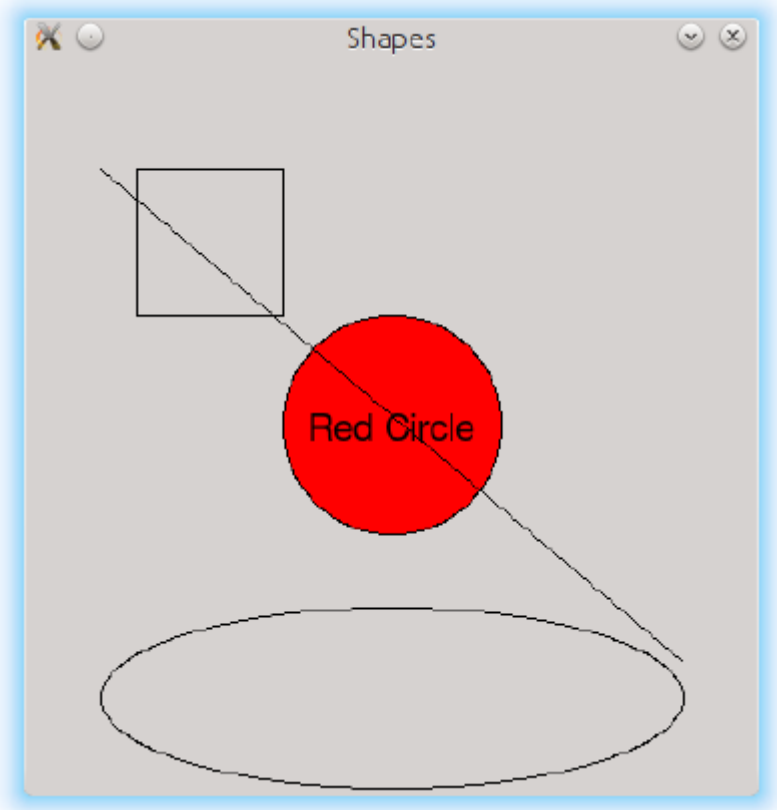
Simple Graphics Programming

```
>>> p = Point(50, 60)
>>> p.getX()
50
>>> p.getY()
60
>>> win = GraphWin()
>>> p.draw(win)
>>> p2 = Point(140, 100)
>>> p2.draw(win)
```



Simple Graphics Programming

```
>>> ### Open a graphics window
>>> win = GraphWin('Shapes')
>>> ### Draw a red circle centered at point
>>> ### (100, 100) with radius 30
>>> center = Point(100, 100)
>>> circ = Circle(center, 30)
>>> circ.setFill('red')
>>> circ.draw(win)
>>> ### Put a textual label in the center of the circle
>>> label = Text(center, "Red Circle")
>>> label.draw(win)
>>> ### Draw a square using a Rectangle object
>>> rect = Rectangle(Point(30, 30), Point(70, 70))
>>> rect.draw(win)
>>> ### Draw a line segment using a Line object
>>> line = Line(Point(20, 30), Point(180, 165))
>>> line.draw(win)
>>> ### Draw an oval using the Oval object
>>> oval = Oval(Point(20, 150), Point(180, 199))
>>> oval.draw(win)
```



Constructor and Objects

LECTURE 4

Graphics Module's Objects

- GraphWin Objects.
- Graphics Objects: Point, Line, Circle, Rectangle, Polygon, Text
- Entry Objects.
- Image Objects.

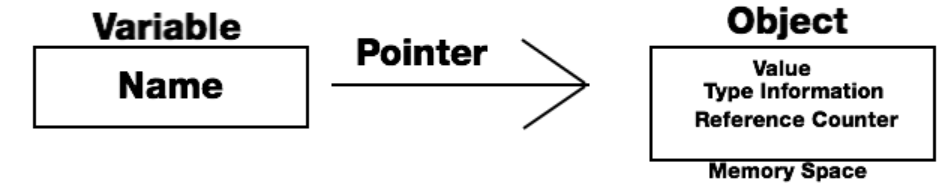
Constructor:

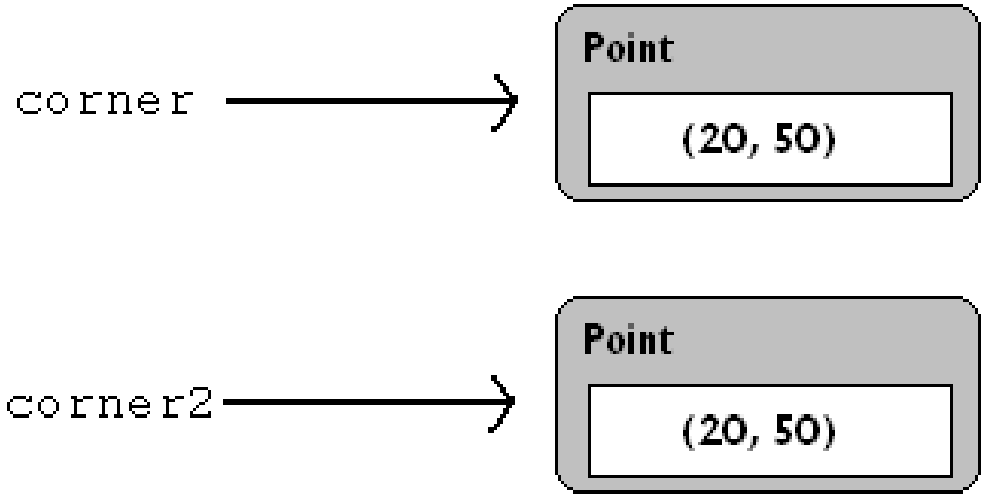
<class-name>(<param1>,<param2>, …)

- To create a new instance of a class, we use a special operation called a constructor. A call to a constructor is an expression that creates a brand new object. The syntax is:

  <class name> (<param1>, <param2>, …)

- This function create a new object of the class and allow a object name variable to point to the object. Two point object named corner, corner2 are created to point to the same location (20, 50) as right.


Variable
Name
Pointer
Object
Value
Type Information
Reference Counter
Memory Space


corner
Point
(20, 50)
corner2
Point
(20, 50)

Member function or Method

<object>.<method-name>(<param1>, <param2>, ...)

- Member function is used for some other language, while Python use the name Method to call a function defined in a class. The method usually works on the member data defined in the class and return some certain result (may also be null).

```
>>> p = Point(20, 50)
>>> p.getX()
20
>>> p.getY()
50
```

- `getX()` and `getY()` are two methods defined in class Point. The instance p (class Point) is applied with the method `getX()` and `getY()` to get the coordination value of (x, y) for the point p.

Pixel system in the Graphics Window

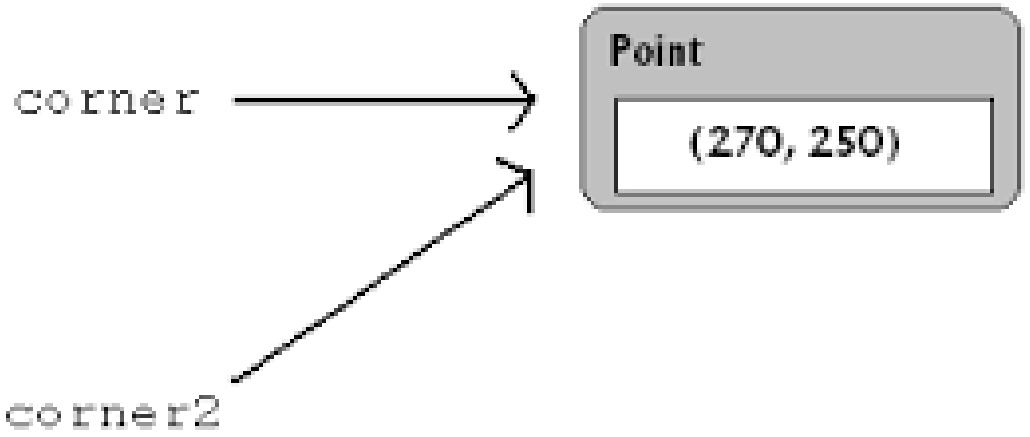
- All of the number is the Point and its member are measured by pixel count, if not otherwise declared. The left-top point of the window is named, `(0, 0),` the origin of the window. The x number grows to the right and y number grows to downward.
- Method: `move(x, y),` move the pixel from the current location to right for x pixel and down for y pixel.
- The following sequence:

```
circ = Circle(Point(100, 100), 30)
win = GraphWin()
circ.draw(win)
```

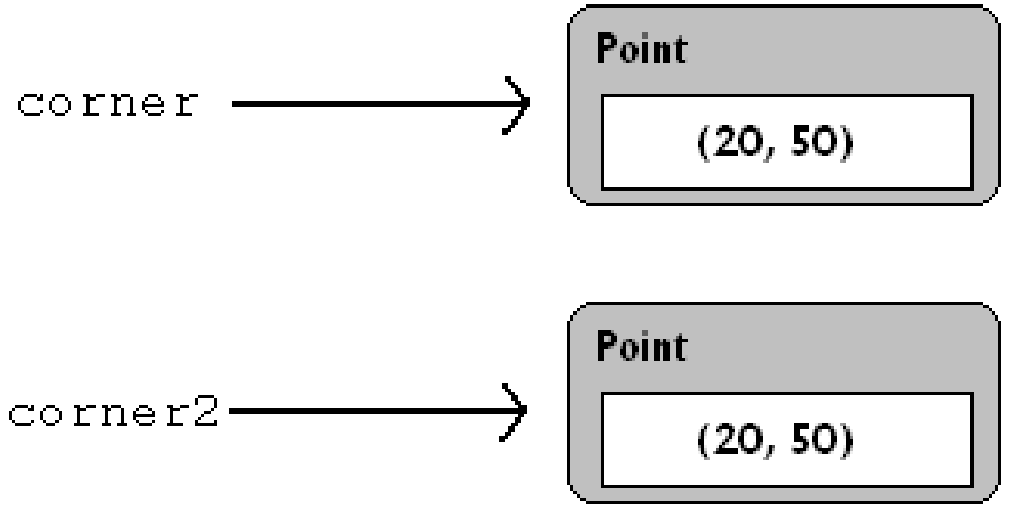
Create an object for a Circle and then draw the circle on a window named win.

Try Example in Page 89 to 91

Aliasing VS Cloning



Alias



Clone

Using of Graphical Objects

LECTURE 5

Using Graphical Objects

- Computation is performed by asking an object to carry out one of its operations.
- In the previous example we manipulated `GraphWin`, `Point`, `Circle`, `Oval`, `Line`, `Text` and `Rectangle`. These are examples of classes.

Using Graphical Objects

- Each object is an **instance** of some class, and the **class** describes the properties of the instance.
- If we say that Augie is a dog, we are actually saying that Augie is a specific individual in the larger **class** of all dogs. Augie is an **instance** of the dog class.

Using Graphical Objects

- To create a new instance of a class, we use a special operation called a constructor.
  `<class-name>(<param1>, <param2>, …)`
- `<class-name>` is the name of the class we want to create a new instance of, e.g. `Circle` or `Point`.
- The parameters are required to initialize the object. For example, `Point` requires two numeric values.

Using Graphical Objects

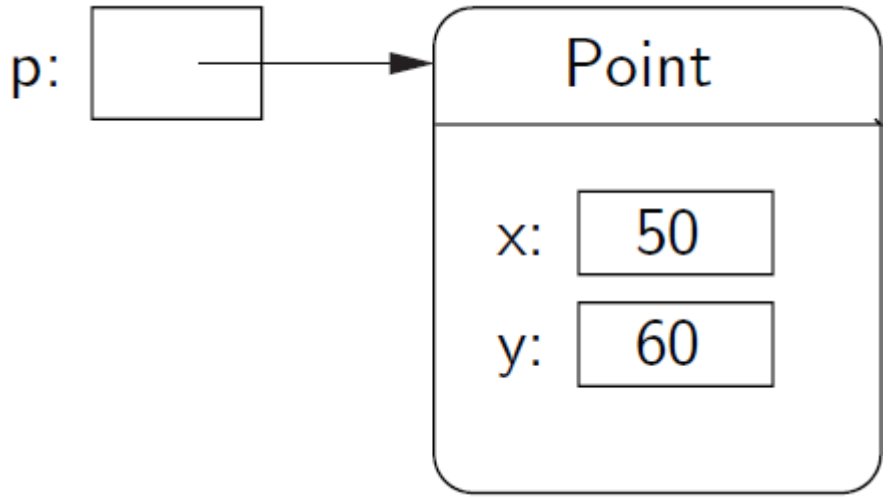
- `p = Point(50, 60)`

  The constructor for the Point class requires two parameters, the x and y coordinates for the point.

- These values are stored as instance variables inside of the object.

Using Graphical Objects

- Only the most relevant instance variables are shown (others include the color, window they belong to, etc.)



Using Graphical Objects

- To perform an operation on an object, we send the object a message. The set of messages an object responds to are called the methods of the object.
- Methods are like functions that live inside the object.
- Methods are invoked using dot-notation:

```
<object>.<method-name>(<param1>, <param2>, …)
```

Using Graphical Objects

- `p.getX()` and `p.getY()` returns the x and y values of the point. Routines like these are referred to as **accessors** because they allow us to access information from the instance variables of the object.

Using Graphical Objects

- Other methods change the **state** of the object by changing the values of the object's instance variables.
- `move(dx, dy)` moves the object dx units in the x direction and dy in the y direction.
- Move erases the old image and draws it in its new position. Methods that change the state of an object are called **mutators**.

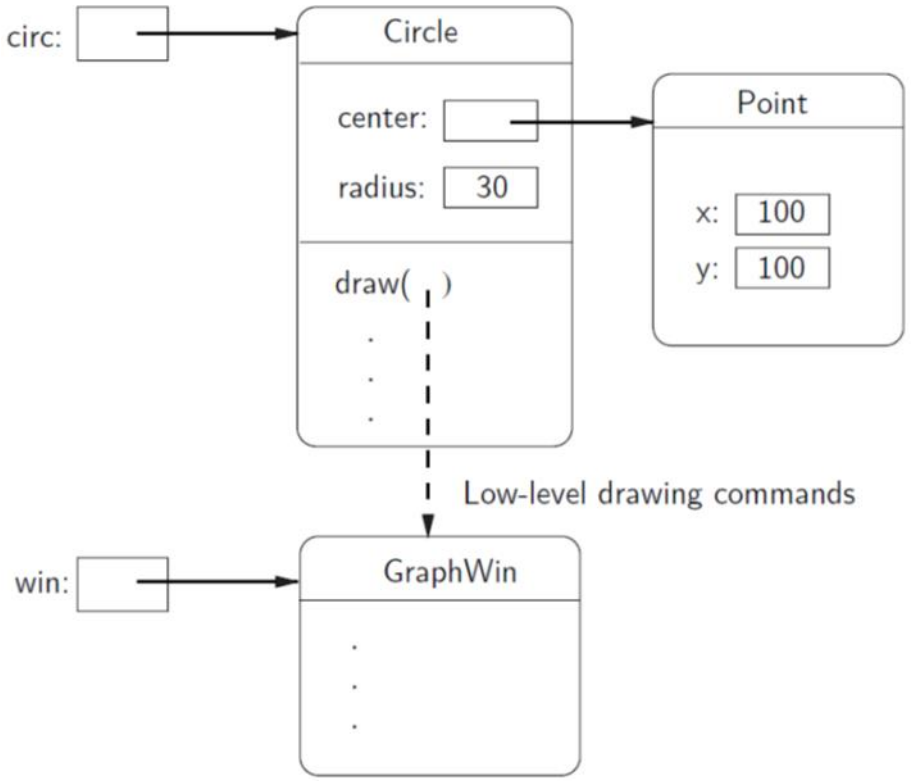
Using Graphical Objects

```
>>> circ = Circle(Point(100, 100), 30)
>>> win = GraphWin()
>>> circ.draw(win)
```

- The first line creates a circle with radius 30 centered at (100,100).
- We used the Point constructor to create a location for the center of the circle.
- The last line is a request to the Circle object circ to draw itself into the GraphWin object win.

Using Graphical Objects

- The draw method uses information about the center and radius of the circle from the instance variable.



Using Graphical Objects

- It's possible for two different variables to refer to the same object – changes made to the object through one variable will be visible to the other.

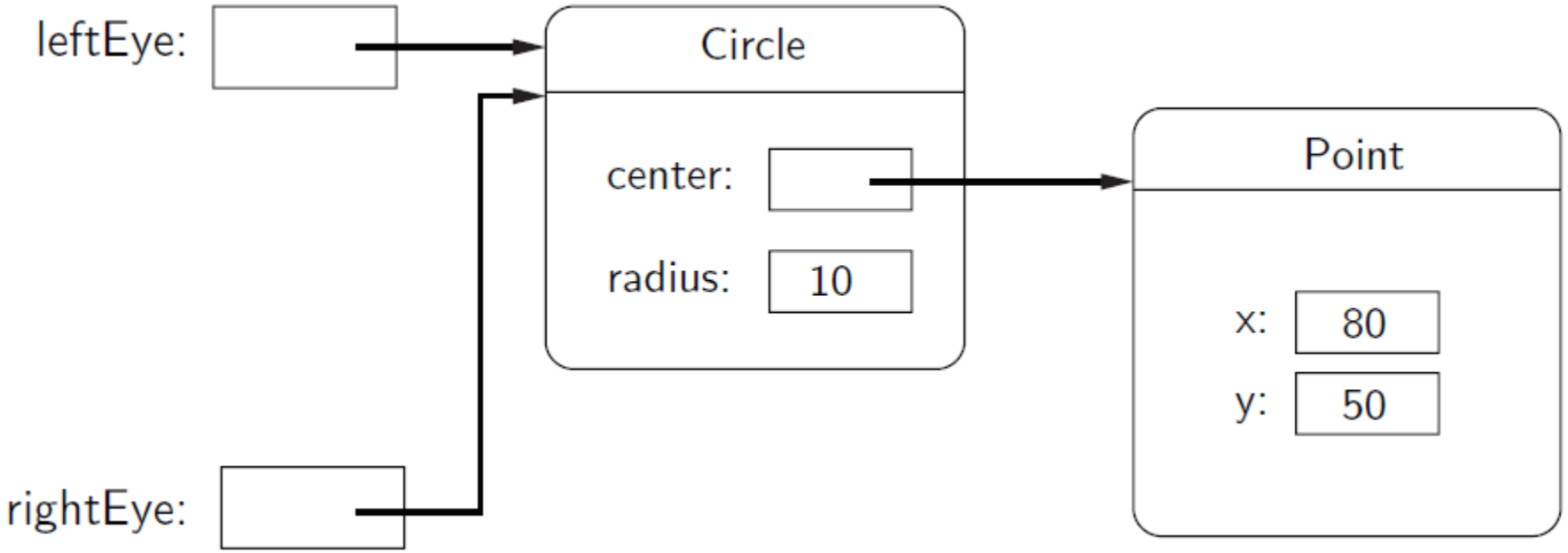
```
>>> leftEye = Circle(Point(80,50), 5)
>>> leftEye.setFill('yellow')
>>> leftEye.setOutline('red')
>>> rightEye = leftEye
>>> rightEye.move(20,0)
```

- The idea is to create the left eye and copy that to the right eye which gets moved over 20 units.

Using Graphical Objects

- The assignment `rightEye = leftEye` makes `rightEye` and `leftEye` refer to the same circle!
- The situation where two variables refer to the same object is called aliasing.

Using Graphical Objects



Using Graphical Objects

- There are two ways to get around this.
- We could make two separate circles, one for each eye:

```
>>> leftEye = Circle(Point(80, 50), 5)
>>> leftEye.setFill('yellow')
>>> leftEye.setOutline('red')
>>> rightEye = Circle(Point(100, 50), 5)
>>> rightEye.setFill('yellow')
>>> rightEye.setOutline('red')
```

Using Graphical Objects

- The graphics library has a better solution. Graphical objects have a clone method that will make a copy of the object!

```
>>> # Correct way to create two circles, using clone
>>> leftEye = Circle(Point(80, 50), 5)
>>> leftEye.setFill('yellow')
>>> leftEye.setOutline('red')
>>> rightEye = leftEye.clone()
>>> # rightEye is an exact copy of the left
>>> rightEye.move(20, 0)
```

Case Study: Future Value

LECTURE 6

Case Study 1: Graphing Future Value

```
# futval.py     /* Program from Chapter 2 , NOW WE WILL PUT THE INPUT AND OUTPUT TO GRAPHIC MODE*/

#  /* This program can be viewed as a model program, we are going to add View program (but still not an object-oriented way */

# A program to compute the value of an investment carried 10 years into the future.

def main():
        print("This program calculates the future value")
        print("of a 10-year investment.")
        principal = eval(input("Enter the initial principal: "))
        apr = eval(input("Enter the annual interest rate: "))
        for i in range(10):
              principal = principal * (1+ apr)
        print("The value in 10 years is: ", principal)
main()
```

Specification for the Graphical Version of futval_graph.py

(1) Print an introduction /* Model */

(2) Get value of principal and apr from user /* View */

(3) Create a GraphWin /* View */

(4) Draw scale labels on left side of window /* View */

(5) Draw bar at position 0 with height corresponding to principal /* View */

(6) For successive years 1 through 10 /* Model */

    Calculate principal = principal * (1 + apr) /* Model */

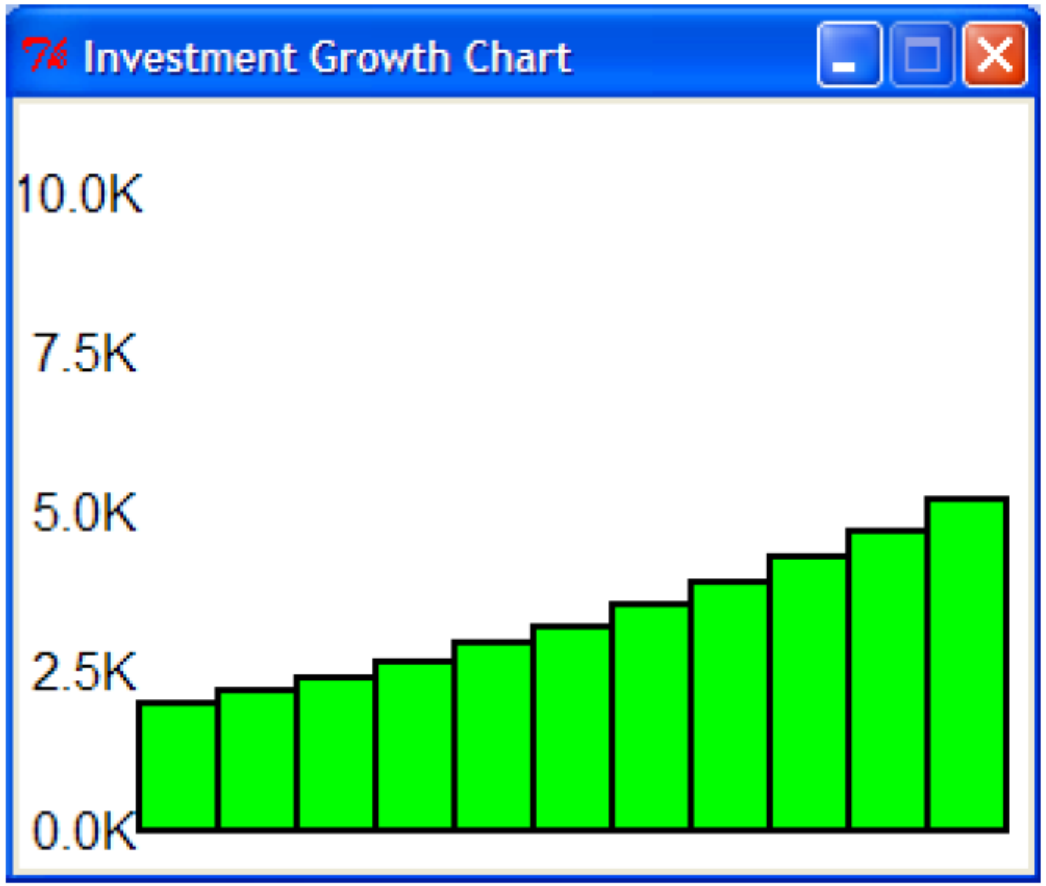
    Draw a bar for this year having a height corresponding to principal /* View */

(7) Wait for user to press Enter /* View */

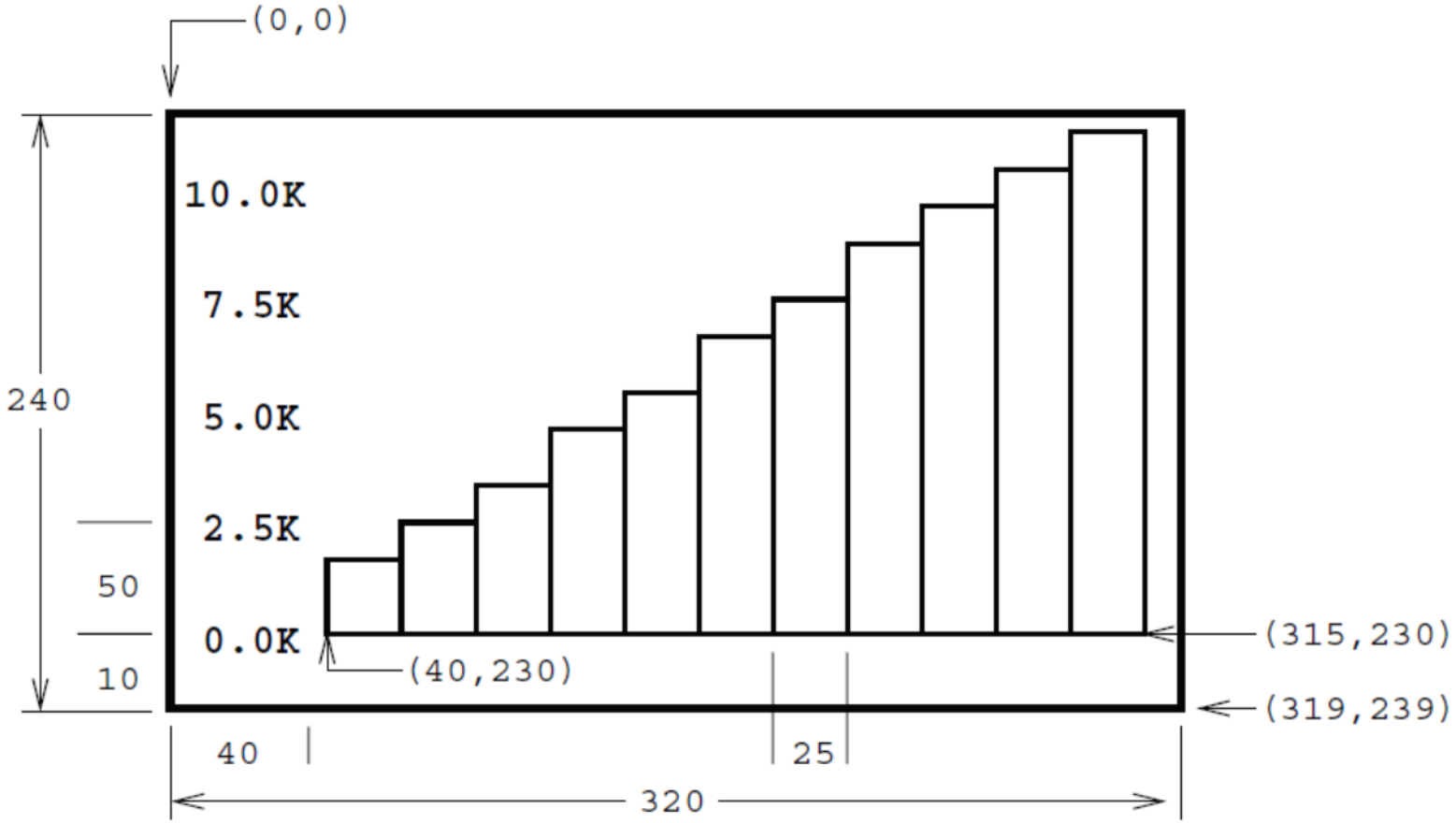
Graphics Operations

- Create a 320x420 (QVGA Format) GraphicWin titled “Investment Growth Chart“
- win = GraphWin(“Investment Growth Chart”, 320, 240)
- Draw scale labels on the left side of window
- Draw label “ 0.0K” at (20, 230)
  Draw label “ 2.5K” at (20, 180)
  Draw label “ 5.0K” at (20, 130)
  Draw label “ 7.5K” at (20, 80)
  Draw label “10.0K” at (20, 30)
- Draw a rectangle from (40, 230) to (65, 230 – principal * 0.02)

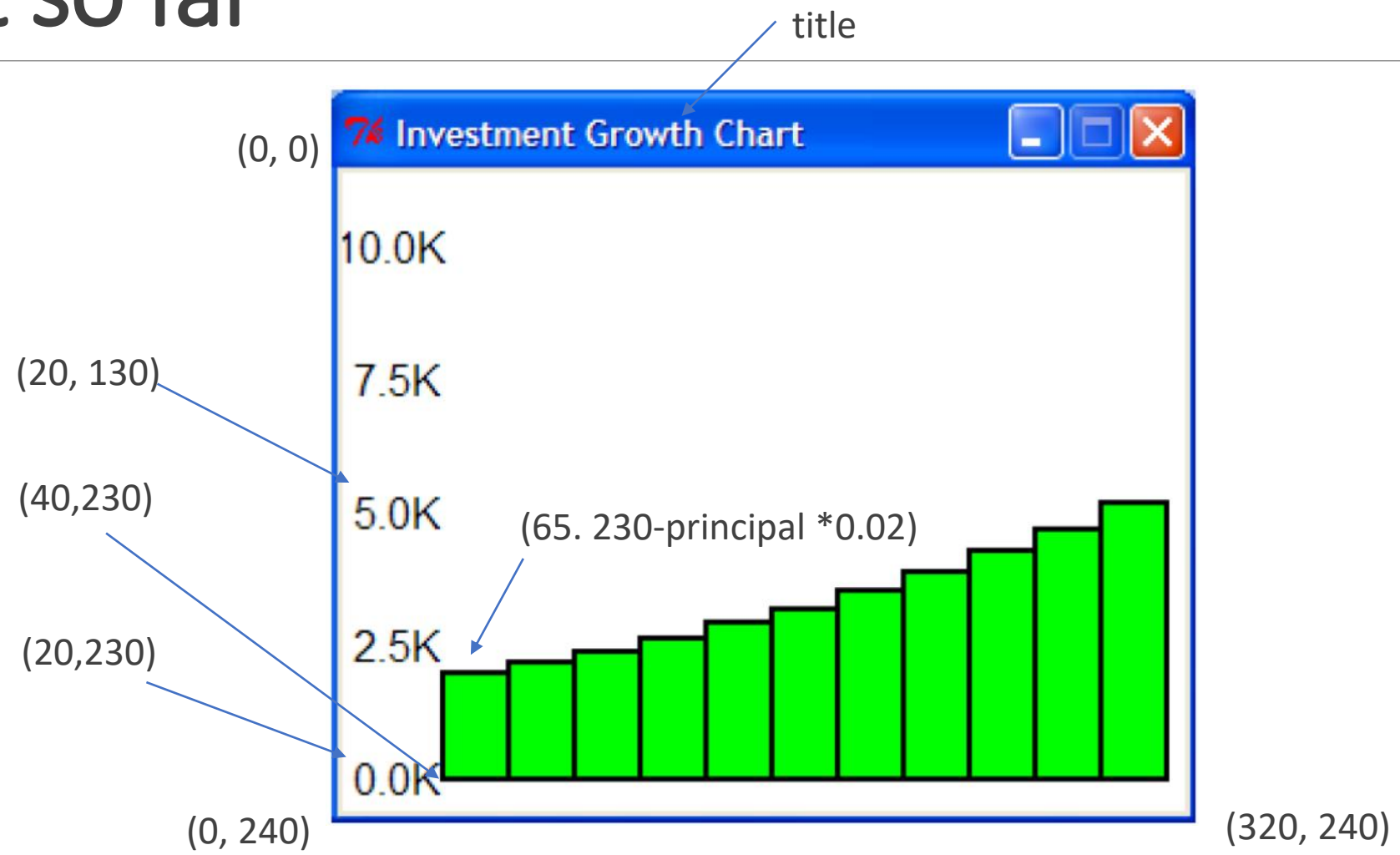
Graphing Future Value/Choosing Coordinates



Graphing Future Value/Choosing Coordinates



Result so far



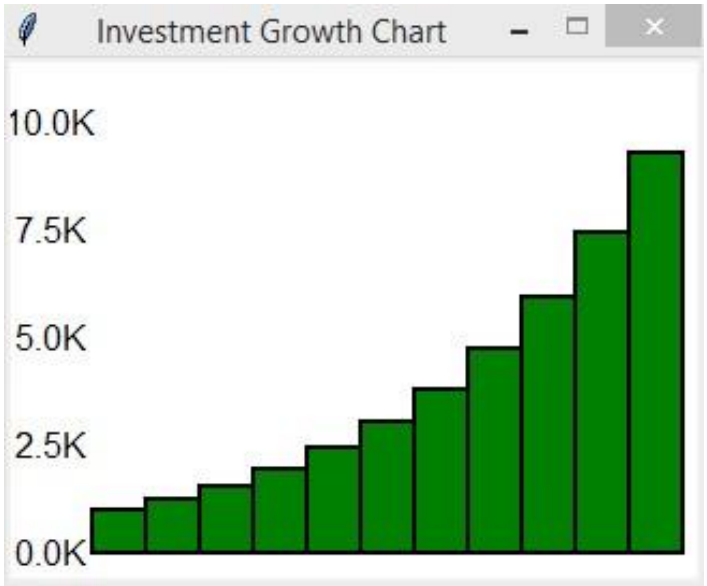
futval_graph.py (I)

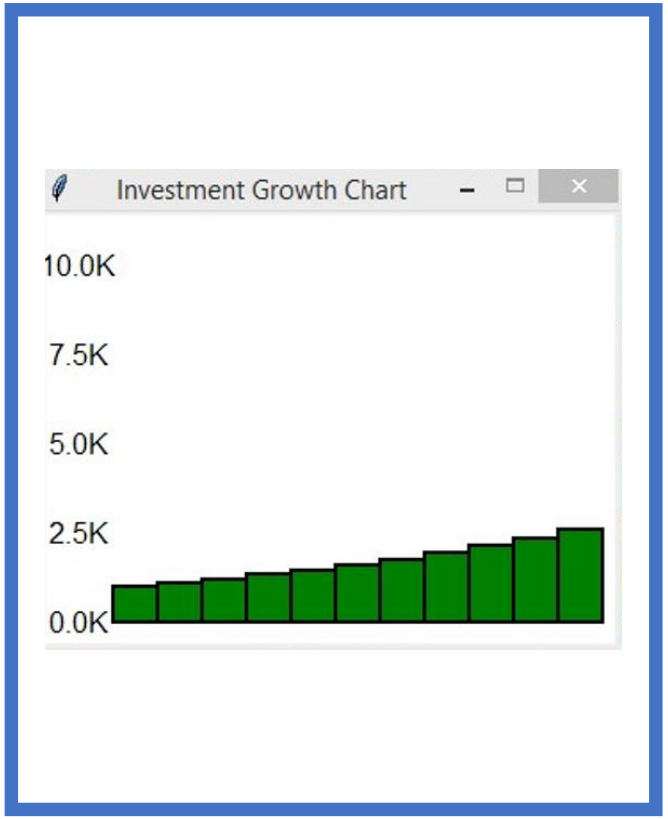
```
# futval_graph.py
from graphics import *
def main():
    #Introduction
    principal = eval(input("Enter the initial principal: "))
    apr = eval(input("Enter the Annualized Interest Rate:"))
    # Create Window
    win = GraphWin("Investment Growth Chart", 320, 240)
    win.setBackground("white")
    Text(Point(20,180), '2.5K').draw(win)
    Text(Point(20,130), '5.0K').draw(win)
    Text(Point(20,80), '7.5K').draw(win)
    Text(Point(20,30), '10.0K').draw(win)
    # Draw Initial bar for Principal
    height = principal * 0.02
    bar = Rectangle(Point(40,230),Point(65, 230-height))
    bar.setFill("green")
    bar.setWidth(2)
    bar.draw(win)
```

futval_graph.py (II)

```
    # Draw bars for successive years
    for year in range(1,11):
        principal = principal * (1+apr)
        xll = year*25+40
        height = principal*0.02
        bar = Rectangle(Point(xll,230),Point(xll+25, 230-height))
        bar.setFill("green")
        bar.setWidth(2)
        bar.draw(win)
    # Close window
    input("Press <Enter> to quit window")
    win.close()
main()
```

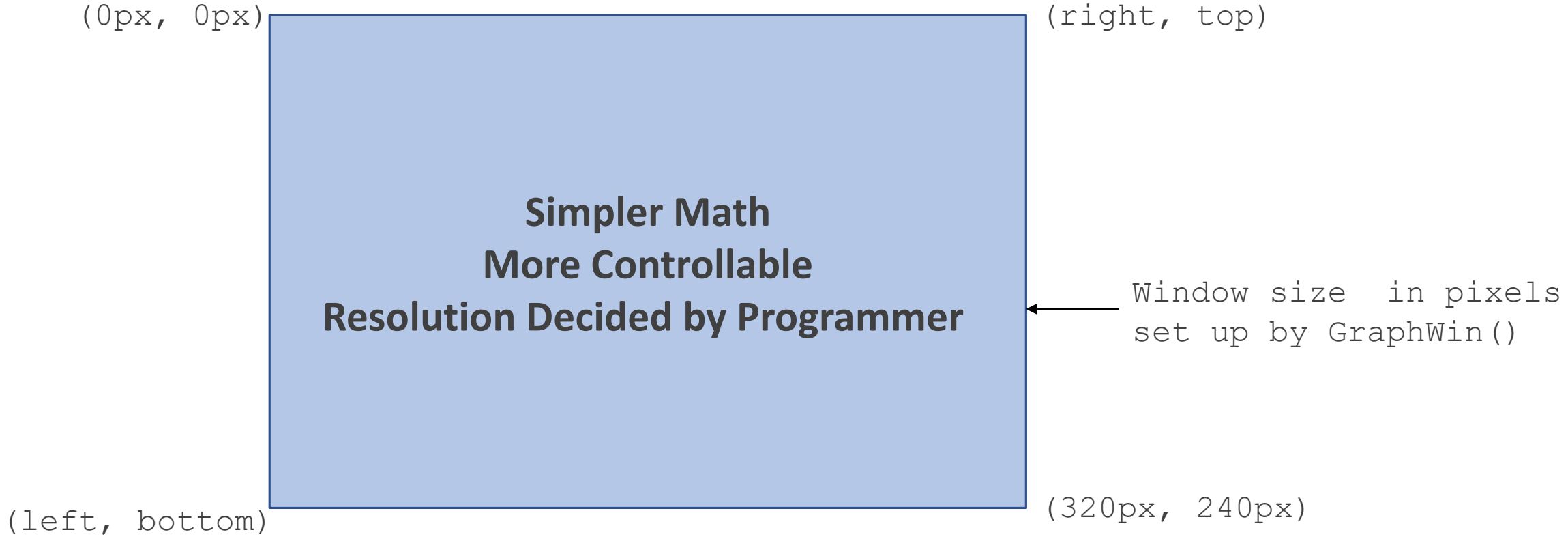
Results (1000, 25%), (1000, 10%)



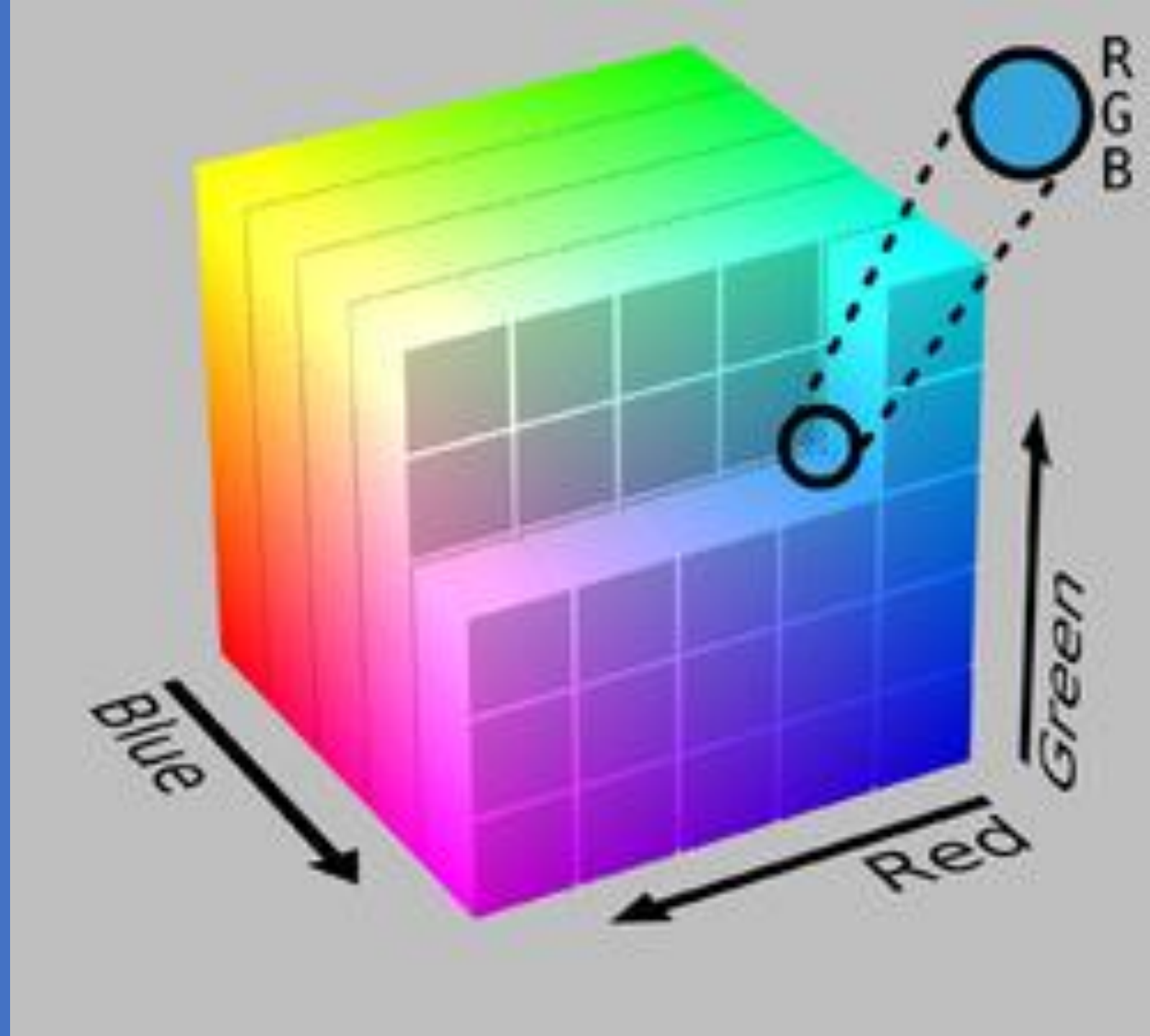


Set Coordinates (avoid pixel calculation)

```
win.setCoords(left, bottom, right, top)    /* window's set Coordinates function */
```



Python Color Name



Red colors

IndianRed	CD 5C 5C	205 92 92
LightCoral	F0 80 80	240 128 128
Salmon	FA 80 72	250 128 114
DarkSalmon	E9 96 7A	233 150 122
LightSalmon	FF A0 7A	255 160 122
Crimson	DC 14 3C	220 20 60
Red	FF 00 00	255 0 0
FireBrick	B2 22 22	178 34 34
DarkRed	8B 00 00	139 0 0

Pink colors

Pink	FF C0 CB	255 192 203
LightPink	FF B6 C1	255 182 193
HotPink	FF 69 B4	255 105 180
DeepPink	FF 14 93	255 20 147
MediumVioletRed	C7 15 85	199 21 133
PaleVioletRed	DB 70 93	219 112 147

Orange colors

LightSalmon	FF A0 7A	255 160 122
Coral	FF 7F 50	255 127 80
Tomato	FF 63 47	255 99 71
OrangeRed	FF 45 00	255 69 0
DarkOrange	FF 8C 00	255 140 0
Orange	FF A5 00	255 165 0

Yellow colors

Gold	FF D7 00	255 215 0
Yellow	FF FF 00	255 255 0
LightYellow	FF FF E0	255 255 224
LemonChiffon	FF FA CD	255 250 205
LightGoldenrodYellow	FA FA D2	250 250 210
PapayaWhip	FF EF D5	255 239 213
Moccasin	FF E4 B5	255 228 181
PeachPuff	FF DA B9	255 218 185
PaleGoldenrod	EE E8 AA	238 232 170
Khaki	F0 E6 8C	240 230 140
DarkKhaki	BD B7 6B	189 183 107

Purple colors

Lavender	E6 E6 FA	230 230 250
Thistle	D8 BF D8	216 191 216
Plum	DD A0 DD	221 160 221
Violet	EE 82 EE	238 130 238
Orchid	DA 70 D6	218 112 214
Fuchsia	FF 00 FF	255 0 255
Magenta	FF 00 FF	255 0 255
MediumOrchid	BA 55 D3	186 85 211
BlueViolet	8A 2B E2	138 43 226
DarkViolet	94 00 D3	148 0 211
DarkOrchid	99 32 CC	153 50 204
DarkMagenta	8B 00 8B	139 0 139
Purple	80 00 80	128 0 128
Indigo	4B 00 82	75 0 130
SlateBlue	6A 5A CD	106 90 205
DarkSlateBlue	48 3D 8B	72 61 139
MediumSlateBlue	7B 68 EE	123 104 238

Brown colors

Cornsilk	FF F8 DC	255 248 220
BlanchedAlmond	FF EB CD	255 235 205
Bisque	FF E4 C4	255 228 196
NavajoWhite	FF DE AD	255 222 173
Wheat	F5 DE B3	245 222 179
BurlyWood	DE B8 87	222 184 135
Tan	D2 B4 8C	210 180 140
RosyBrown	BC 8F 8F	188 143 143
SandyBrown	F4 A4 60	244 164 96
Goldenrod	DA A5 20	218 165 32
DarkGoldenrod	B8 86 0B	184 134 11
Peru	CD 85 3F	205 133 63
Chocolate	D2 69 1E	210 105 30
SaddleBrown	8B 45 13	139 69 19
Sienna	A0 52 2D	160 82 45
Brown	A5 2A 2A	165 42 42
Maroon	80 00 00	128 0 0

Blue/Cyan colors

Aqua	00 FF FF	0 255 255
Cyan	00 FF FF	0 255 255
LightCyan	E0 FF FF	224 255 255
PaleTurquoise	AF EE EE	175 238 238
Aquamarine	7F FF D4	127 255 212
Turquoise	40 E0 D0	64 224 208
MediumTurquoise	48 D1 CC	72 209 204
DarkTurquoise	00 CE D1	0 206 209
CadetBlue	5F 9E A0	95 158 160
SteelBlue	46 82 B4	70 130 180
LightSteelBlue	B0 C4 DE	176 196 222
PowderBlue	B0 E0 E6	176 224 230
LightBlue	AD D8 E6	173 216 230
SkyBlue	87 CE EB	135 206 235
LightSkyBlue	87 CE FA	135 206 250
DeepSkyBlue	00 BF FF	0 191 255
DodgerBlue	1E 90 FF	30 144 255
CornflowerBlue	64 95 ED	100 149 237
MediumSlateBlue	7B 68 EE	123 104 238
RoyalBlue	41 69 E1	65 105 225
MediumBlue	00 00 CD	0 0 205
DarkBlue	00 00 8B	0 0 139
Navy	00 00 80	0 0 128
MidnightBlue	19 19 70	25 25 112

Green colors

GreenYellow	AD FF 2F	173 255 47
Chartreuse	7F FF 00	127 255 0
LawnGreen	7C FC 00	124 252 0
Lime	00 FF 00	0 255 0
LimeGreen	32 CD 32	50 205 50
PaleGreen	98 FB 98	152 251 152
LightGreen	90 EE 90	144 238 144
MediumSpringGreen	00 FA 9A	0 250 154
SpringGreen	00 FF 7F	0 255 127
MediumSeaGreen	3C B3 71	60 179 113
SeaGreen	2E 8B 57	46 139 87
ForestGreen	22 8B 22	34 139 34
Green	00 80 00	0 128 0
DarkGreen	00 64 00	0 100 0
YellowGreen	9A CD 32	154 205 50
OliveDrab	6B 8E 23	107 142 35
Olive	80 80 00	128 128 0
DarkOliveGreen	55 6B 2F	85 107 47
MediumAquamarine	66 CD AA	102 205 170
DarkSeaGreen	8F BC 8F	143 188 143
LightSeaGreen	20 B2 AA	32 178 170
DarkCyan	00 8B 8B	0 139 139
Teal	00 80 80	0 128 128

White colors

White	FF FF FF	255 255 255
Snow	FF FA FA	255 250 250
Honeydew	F0 FF F0	240 255 240
MintCream	F5 FF FA	245 255 250
Azure	F0 FF FF	240 255 255
AliceBlue	F0 F8 FF	240 248 255
GhostWhite	F8 F8 FF	248 248 255
WhiteSmoke	F5 F5 F5	245 245 245
Seashell	FF F5 EE	255 245 238
Beige	F5 F5 DC	245 245 220
OldLace	FD F5 E6	253 245 230
FloralWhite	FF FA F0	255 250 240
Ivory	FF FF F0	255 255 240
AntiqueWhite	FA EB D7	250 235 215
Linen	FA F0 E6	250 240 230
LavenderBlush	FF F0 F5	255 240 245
MistyRose	FF E4 E1	255 228 225

Gray colors

Gainsboro	DC DC DC	220 220 220
LightGrey	D3 D3 D3	211 211 211
Silver	C0 C0 C0	192 192 192
DarkGray	A9 A9 A9	169 169 169
Gray	80 80 80	128 128 128
DimGray	69 69 69	105 105 105
LightSlateGray	77 88 99	119 136 153
SlateGray	70 80 90	112 128 144
Black	00 00 00	0 0 0

Interactive Graphics

LECTURE 7

Interactive Graphics

- In a GUI environment, users typically interact with their applications by clicking on buttons, choosing items from menus, and typing information into on-screen text boxes.
- Event-driven programming draws interface elements (widgets) on the screen and then waits for the user to do something.

Interactive Graphics

- An event is generated whenever a user moves the mouse, clicks the mouse, or types a key on the keyboard.
- An event is an object that encapsulates information about what just happened.
- The event object is sent to the appropriate part of the program to be processed, for example, a button event.

Interactive Graphics

- The graphics module hides the underlying, low-level window management and provides two simple ways to get user input in a `GraphWin`.

Getting Mouse Clicks

LECTURE 8

Getting Mouse Clicks

- We can get graphical information from the user via the `getMouse` method of the `GraphWin` class.
- When `getMouse` is invoked on a `GraphWin`, the program pauses and waits for the user to click the mouse somewhere in the window.
- The spot where the user clicked is returned as a `Point`.

Getting Mouse Clicks

- The following code reports the coordinates of a mouse click:

```
from graphics import *
win = GraphWin("Click Me!")
p = win.getMouse()
print("You clicked", p.getX(), p.getY())
```

- We can use the accessors like `getX` and `getY` or other methods on the point returned.

Case Study II: click.py

```
# click.py
from graphics import *
def main():
    win = GraphWin("Click Me!")
    for i in range(10):
        p=win.getMouse()
        print("You Click at:", p.getX(), p.getY())

main()
```

Triangle

LECTURE 9

Case Study III: triangle.pyw

```
# triangle.pyw
# Interactive graphics program to draw a triangle

from graphics import *

def main():
    win = GraphWin("Draw a Triangle")
    win.setCoords(0.0, 0.0, 10.0, 10.0)
    message = Text(Point(5, 0.5), "Click on three points")
    message.draw(win)

    # Get and draw three vertices of triangle
    p1 = win.getMouse()
    p1.draw(win)
    p2 = win.getMouse()
    p2.draw(win)
    p3 = win.getMouse()
    p3.draw(win)
```

Getting Mouse Clicks

```
# Use Polygon object to draw the triangle
    triangle = Polygon(p1,p2,p3)
    triangle.setFill("peachpuff")
    triangle.setOutline("cyan")
    triangle.draw(win)

    # Wait for another click to exit
    message.setText("Click anywhere to quit.")
    win.getMouse()

main()
```

Results of triangle.py

Getting Mouse Clicks

- Notes:
  - If you are programming in a windows environment, using the .pyw extension on your file will cause the Python shell window to not display when you double-click the program icon.
  - There is no triangle class. Rather, we use the general Polygon class, which takes any number of points and connects them into a closed shape.

Getting Mouse Clicks

- Once you have three points, creating a triangle polygon is easy:

```
triangle = Polygon(p1, p2, p3)
```

- A single text object is created and drawn near the beginning of the program.

```
message = Text(Point(5,0.5), "Click on three
points")
message.draw(win)
```

- To change the prompt, just change the text to be displayed.

```
message.setText("Click anywhere to quit.")
```

Handling Textual Input

LECTURE 10

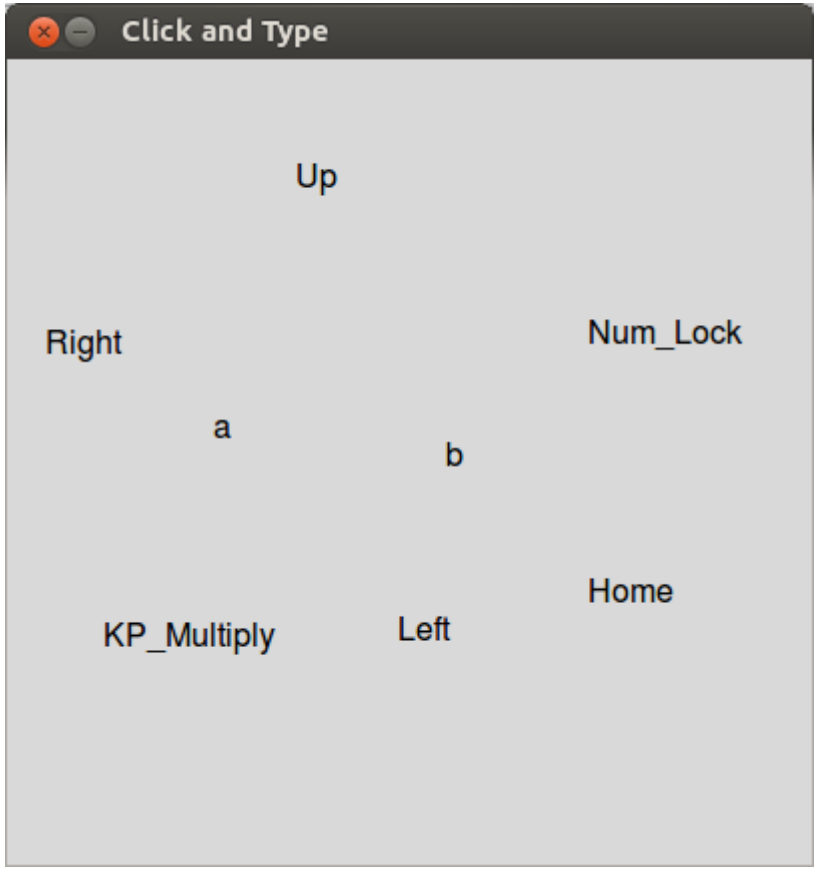
Handling Textual Input

- The triangle program's input was done completely through mouse clicks.
- The `GraphWin` object provides a `getKey()` method that works like the `getMouse` method.

Handling Textual Input

```
# clickntype.py
from graphics import *
def main():
    win = GraphWin("Click and Type", 400,
400)
    for i in range(10):
        pt = win.getMouse()
        key = win.getKey()
        label = Text(pt, key)
        label.draw(win)
```

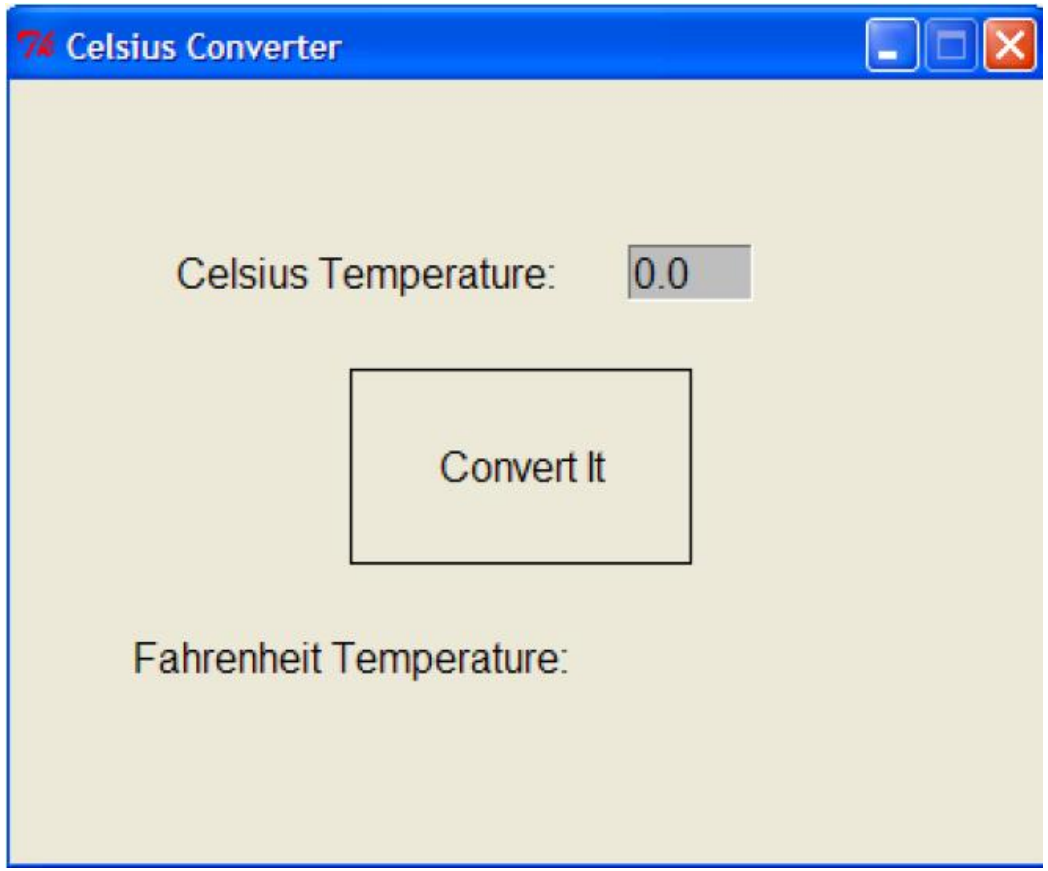
Handling Textual Input



Handling Textual Input

- There's also an Entry object that can get keyboard input.
- The Entry object draws a box on the screen that can contain text. It understands setText and getText, with one difference that the input can be edited.

Handling Textual Input

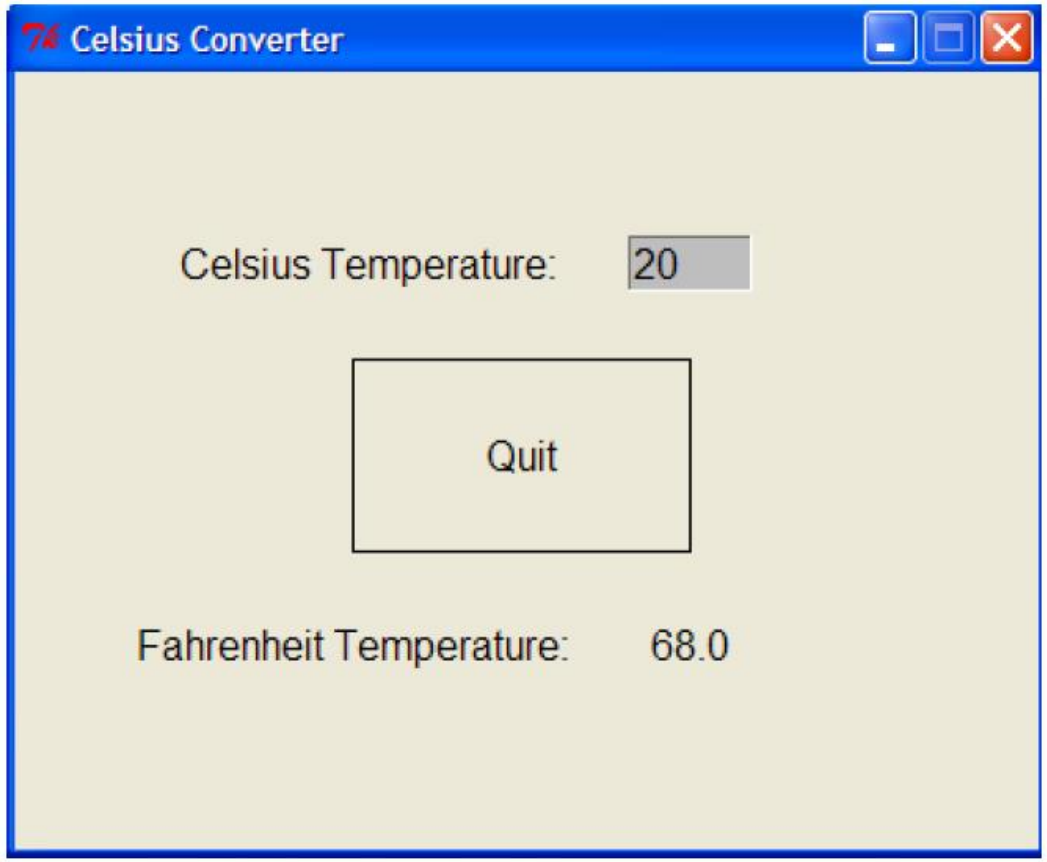


```
# convert_gui.pyw
# Program to convert Celsius to Fahrenheit using a simple
#   graphical interface.
from graphics import *
def main():
    win = GraphWin("Celsius Converter", 300, 200)
    win.setCoords(0.0, 0.0, 3.0, 4.0)

    # Draw the interface
    Text(Point(1,3), "   Celsius Temperature:").draw(win)
    Text(Point(1,1), "Fahrenheit Temperature:").draw(win)
    input = Entry(Point(2,3), 5)
    input.setText("0.0")
    input.draw(win)
    output = Text(Point(2,1),"")
    output.draw(win)
    button = Text(Point(1.5,2.0),"Convert It")
    button.draw(win)
    Rectangle(Point(1,1.5), Point(2,2.5)).draw(win)
```

Handling Textual Input

```
    # wait for a mouse click
    win.getMouse()
    # convert input
    celsius = eval(input.getText())
    fahrenheit = 9.0/5.0 * celsius + 32
    # display output and change button
    output.setText(fahrenheit)
    button.setText("Quit")
    # wait for click and then quit
    win.getMouse()
    win.close()

main()
```

Handling Textual Input



Handling Textual Input

- When run, this program produces a window with an entry box for typing in the Celsius temperature and a button to "do" the conversion.
  - The button is for show only! We are just waiting for a mouse click anywhere in the window.

Handling Textual Input

- Initially, the input entry box is set to contain “0.0”.
- The user can delete this value and type in another value.
- The program pauses until the user clicks the mouse – we don’t care where so we don’t store the point!

Handling Textual Input

- The input is processed in three steps:
  - The value entered is converted into a number with float.
  - This number is converted to degrees Fahrenheit.
  - This number is then converted to a string and formatted for display in the output text area.

Homework

LECTURE 11

Homework 4

1. Download the Reference article for graphics.py:
2. http://mcsp.wartburg.edu/zelle/python/graphics/graphics.pdf
3. Work on Exercise True/False, Multiple Choice, Discussion, Program Exercise 1,2 3, 9