

CSI 31 Lecture 8

Topics: *Chapter 4. Objects and Graphics*

4.1 Overview

4.2 The object of objects

4.3 Simple graphical programming

4.4 Using Graphical Objects

link to the graphics library:

<http://mcsp.wartburg.edu/zelle/python/>

4.1 Overview

4.2 The Object of Objects

So far we used built-in Python data types for our programs (*int*, *float*, *long*, *str*).

Each type:

- Can represent the certain set of data values, and
- Has a set of associated operations.

Data – *Passive* entities, they are manipulated and combined via *active* operations.

4.1 Overview

4.2 The Object of Objects

Another (**modern**) **approach**:

programs are built using **Object Oriented (OO)** approach.

The basic idea:

we view a complex system as an interaction of simpler **objects**.

OO objects:

- **Contain data** (*know staff*). Also called “attributes”.
- **Have operations** (*can do staff*). Also called “methods”.

For example, a car is an object. It has some attributes (its brand, its color, its model, its length,...) and can do things (can interact with a human to be driven, can move, can crash, can break down,...)

A student is an object (in python, of course). What are some attributes? What about methods?

4.1 Overview

4.2 The Object of Objects

Another (modern) approach:

programs are built using Object Oriented (OO) approach.

The basic idea:

we view a complex system as an interaction of simpler objects.

OO objects:

- *Contain data* (*know staff*). Also called “attributes”.
- *Have operations* (*can do staff*). Also called “methods”.

For example, a car is an object. It has some attributes (its brand, its color, its model, its length,...) and can do things (can interact with a human to be driven, can move, can crash, can break down,...)

A student is an object (in python, of course). What are some attributes? What about methods?

We will use graphics to show the object oriented approach.

4.3 Simple Graphics Programming

Instructions for getting the graphics library:

We will use the library written specifically for our book: [graphics.py](#)

You can download it from here:

<http://mcsp.wartburg.edu/zelle/python/>

Put/copy it into the folder 'Lib' in the Python's folder.

Note: maybe this library is already in your system. Check first by writing “import graphics” into the prompt.

The library is already available in the lab

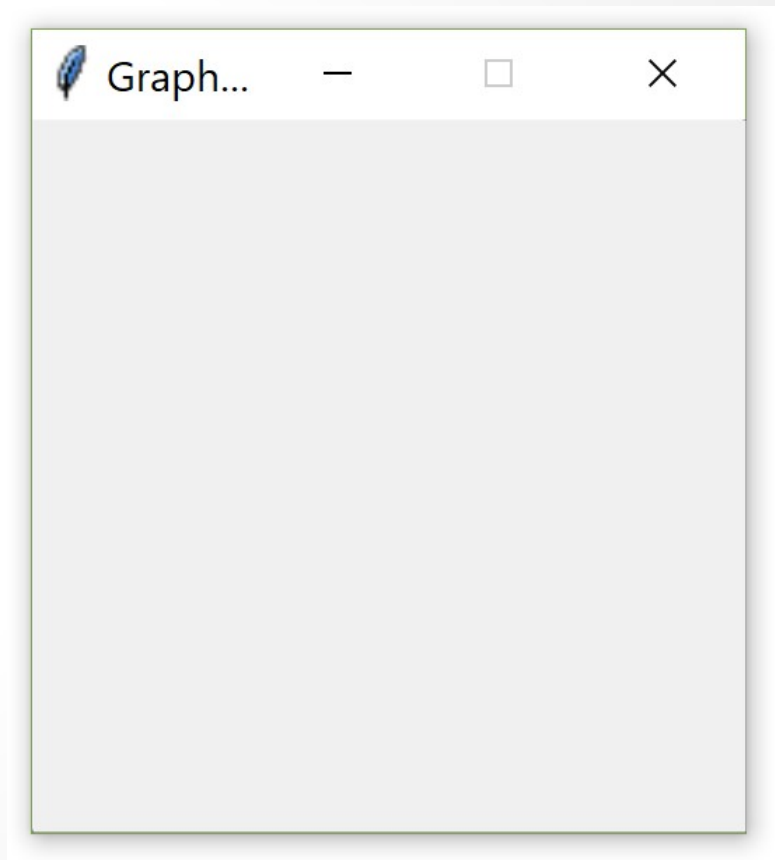
4.3 Simple Graphics Programming

Type in the following in the interactive window:

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

- This will create a new object with the name 'win'

You can see that there is a new window called '**Graphics Window**' (if you re-size it)



4.3 Simple Graphics Programming

Type in the following in the interactive window:

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

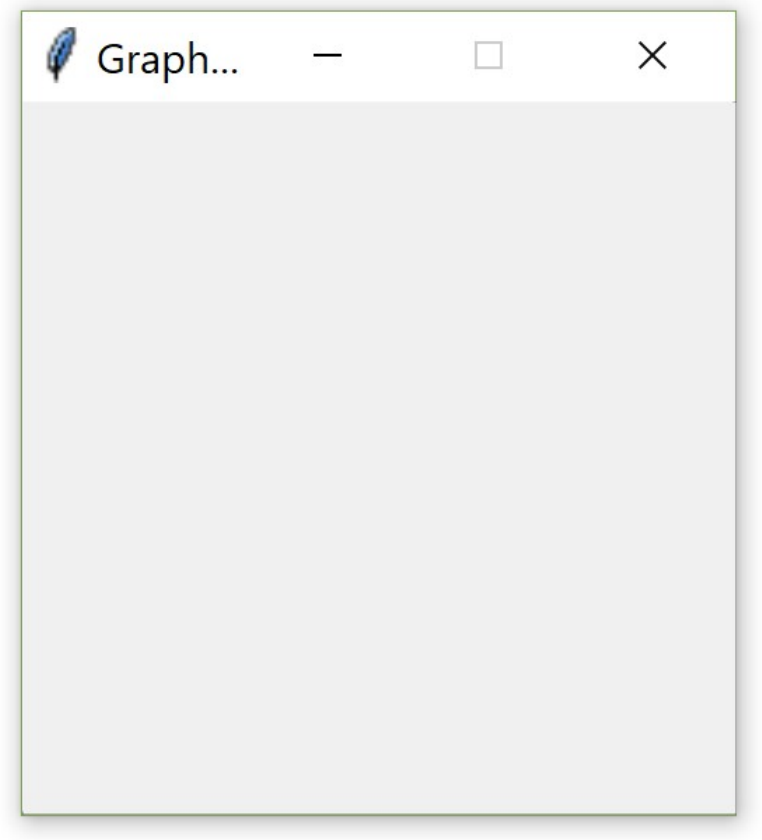
- This will create a new object with the name 'win'

You can see that there is a new window called '**Graphics Window**' (if you re-size it)

Type in:

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

- The object 'win' is destroyed and we won't see that Graphics Window anymore.



4.3 Simple Graphics Programming

We will be working with lots of commands (functions) from graphics library, let's do an alternative import of that library:

```
>>> from graphics import *
```

- It means *'load all the operations/commands and constants from the library module graphics'*
- The imported commands become directly available, without the dot-notation.

Now we can work more comfortably:

```
>>> win = GraphWin()
```


4.3 Simple Graphics Programming

GraphWin()

```
GraphWin(title="Graphics window",width=200,  
height=200, autoflush=True)
```

4.3 Simple Graphics Programming

GraphWin()

```
GraphWin(title="Graphics window",width=200,  
height=200, autoflush=True)
```

by default, the size of the window created is 200 pixels × 200 pixels
the title of the window is "Graphics Window" and
all the changes in it are automatically displayed

See example program [example1.py](#)

4.3 Simple Graphics Programming

pixels

Pixels (picture elements)– are tiny points on our displays. Each of them has color.

By controlling the color of each pixel we can control what is displayed on the screen.

The position of each pixel is a pair (x,y) :
 x-coordinate and **y**-coordinate.

4.3 Simple Graphics Programming

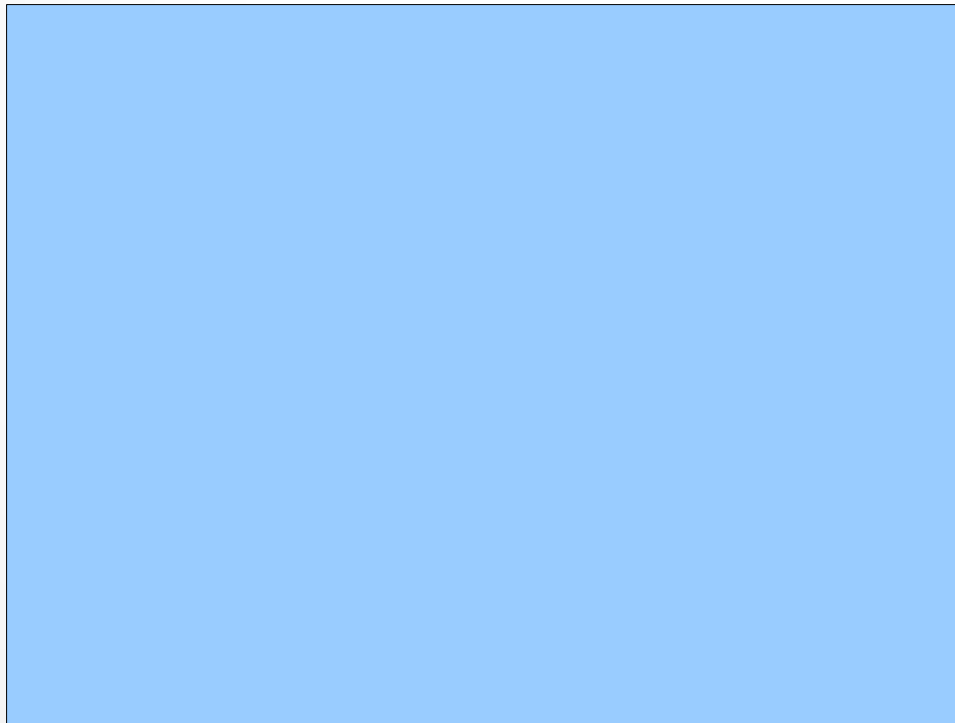
Upper left corner : (0,0),

Lower right corner:

(max_vertical_resolution, max_horizontal_resolution)

Origin of our "rectangular coordinate system" is at the top-left corner.

Origin (0,0)



See example program [points.py](#)

4.4 Using Graphical Objects

The module provides the following drawable objects: `Point`, `Line`, `Circle`, `Oval`, `Rectangle`, `Polygon`, and `Text`. Each of these kinds are examples of `classes`.

When we write an assignment `p1=Point(10,30)`, the following happens:

An *instance* of class `Point` is created (called object), and is assigned to variable `p1`.

In over words, we never «use» classes themselves, we create instances of classes (objects) and work with them.

4.4 Using Graphical Objects

Every object is an instance of some class, and the class describes the properties the instance has.

4.4 Using Graphical Objects

To define a class we

1. Define a **constructor(s)**
(expression that creates instance of this class)
2. Define **variable(s) (attributes)**
3. Define **method(s) (function(s))**

4.4 Using Graphical Objects

1. defining a constructor(s)

constructor is an expression that creates brand new object.

The general form is: `<class-name>(<param1>, <param2>, ...)`

name of class
(Circle or Point or ...)

parameters that are
required to initialize the
object

4.4 Using Graphical Objects

1. defining a constructor(s)

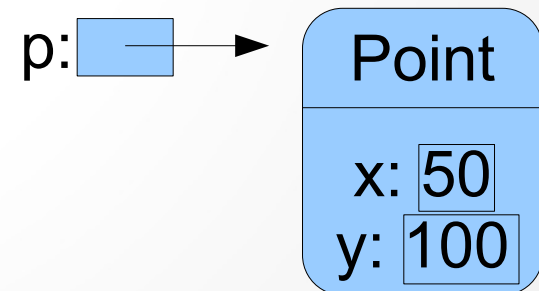
constructor is an expression that creates brand new object.

The general form is: `<class-name>(<param1>, <param2>, ...)`

Example:

```
p = Point(50,100)
```

- the class name is **Point**, and here is a call to the constructor with two parameters (x-coordinate, y-coordinate)



4.4 Using Graphical Objects

1. defining a constructor(s)

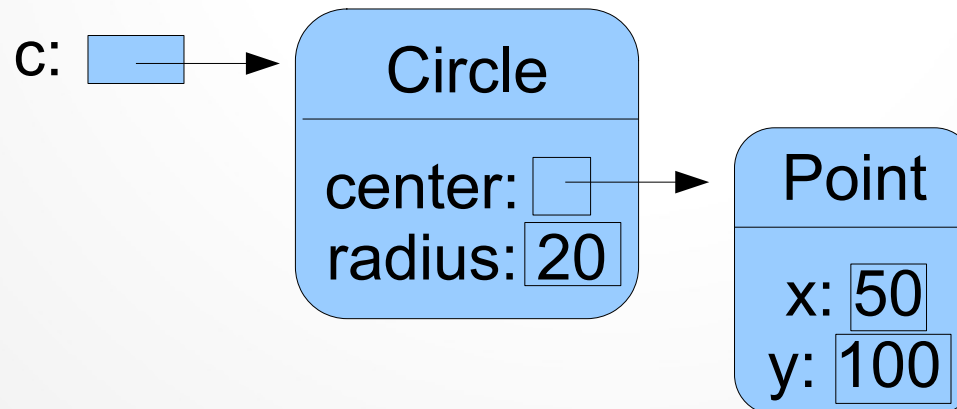
constructor is an expression that creates brand new object.

The general form is: `<class-name>(<param1>, <param2>, ...)`

Example:

```
c = Circle(Point(50,100),20)
```

- the class name is `Circle`, the constructor needs two parameters: center (should be of type `Point`), and radius.



4.4 Using Graphical Objects

2. define `variable(s)`

In the `Point` class, `x`-coordinate value and `y`-coordinate value are stored as *instance variables* inside of the object.

4.4 Using Graphical Objects


3. define **method(s)** (function(s))

To perform an operation on an object we use one of the methods of the object:

Example:

```
p1=Point(30,30)  
p1.draw(win)
```

calls a method
«draw» of object p1



«draw» is a function (method) of class **Point**, that draws a point. It requires one parameter (where to draw the object, in this case in the window “win”).

The general form of a method call:

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

4.4 Using Graphical Objects

3. define `method(s)` (function(s))

Methods can be without parameters (when they are not needed):
`p1.getX()`

Methods like this, that allow us to access information from the instance variables of the objects are called *accessors*.

4.4 Using Graphical Objects

3. define `method(s)` (function(s))

Methods can change the values of an object's instance variables, hence changing the `state` of an object:

`p1.move(10, 30)` – moves the point 10 pixels to the right and 30 pixels down.

Such methods are called *mutators*.

The general form of the move method is `move(dx, dy)`.
All the graphical objects have this method.