Class: 9/21/2021

What is an object?

So far, we've been doing procedural programming:

↳ data is passive → manipulated / combined with operations

... but this doesn't reflect real life!

[Objects] super variables!

- contains data called instance variables (knows stuff)
- has its own functions called methods (can do stuff)
- interacts w/ other objects
- [class] - the class is like the definition of the object
  - ↳ determines the instance variables / methods that define a type of an object

ex) · class Dog ← creating the "recipe" to a cake. Doesn't exist as an object yet

· Chloe = Dog(...)

↑ an object, or an [instance] of the Dog class

↑ the object contains all the data about a specific dog

So creating an object is like creating a variable!

· Dog class could contain instance variables for --

name, breed, gender, age, owner, etc.

myDog = Dog("Chloe", "Westie", 14)

instance variable

creating a new class

↳ instance variable values as parameters

· methods:

the period means possession - the method that belongs to the variable

variable.method(parameters)

ex) myDog.eat("Treat")

· interact w/ other objects → Dog object can interact w/ other objects like other Dogs, Humans, Squirrels, etc.

ex) myDog.play(anotherDog)

some other Dog object — all of its info is jammed into this method.

Chapter 4: Objects and Graphics
9/26/2021

Chapter 4.1: overview

Object oriented (OO) - encompasses a number of principles for designing and implementing software

Graphical user interface (GUI) - provides visual elements

Tkinter → standard python GUI module

↳ we will use a graphics library graphics.py, a wrapper around Tkinter

Chapter 4.2: The Object of Objects

Basic idea behind OO development: view a complex system as the interaction of simpler objects

[OO object] a sort of "active" data type that combines both data and operations

↳ objects know stuff (contain data) and can do stuff (have operations)

↳ objects interact by sending each other "messages", or requests for an object to perform one of its operations

Chapter 4.3: Simple Graphics programming

steps: 1) import graphics ← import the graphics module

2) create a graphics window or GraphWin, a place on the screen where the graphics will appear

↳ win = graphics.GraphWin()

this will create a new window ~~of~~ on your screen; it will have the title "Graphics window."

since GraphWin is an object, assign it to the variable win

↳ now, we can manipulate the window object through this variable

ex) destroy the window when done

win.close() ← note: dot notation, but w/ a variable name, not a module name chapter 4.3 continued:

Alternative process:

1) type "from graphics import *"

from allows you to load specific definitions from a library

can list the names of definitions to be imported, or use the asterisk to import everything defined in the module

2) wow! now we can say:

```
win = GraphWin()   ← instead
```

[pixels] - make up a graphics window; tiny points; short for "picture elements"; controlling their colors controls what is showing in the window

↳ by default: GraphWin is 200×200 pixels

[Point] - simplest object in the graphics module; represents a location in a GraphWin.

↳ (x, y), where x ⇒ horizontal pos, y ⇒ vertical pos.

↳ (0, 0) is in the upper-left corner of the window

↳ x values increase left to right
y values increase top to bottom

↳ drawing a Point sets the color of the corresponding pixel in GraphWin → ☆ black is the default color for drawing

ex)

```
p = Point(50, 60)
p.getX()  ← outputs 50
p.getY()  ← outputs 60
win = GraphWin()
p.draw(win)
P2 = Point(140, 100)
P2.draw(win)
```

2 points drawn into a window using the draw operation

Chapter 4.3 continued:

the graphics library contains commands for drawing lines, circles, rectangles, ovals, polygons, and text

ex) win = GraphWin('shapes') ← changes the window title → an optional parameter!

[circle]

```
center = Point(100,100)
circ = Circle(center, 30)
circ.setFill('red')
circ.draw(win)
```

[Textual label]

```
label = Text(center, "Red circle")
label.draw(win)
```

[square]

```
rect = Rectangle(Point(30,30), Point(70,70))
rect.draw(win)
```

[line]

```
line = Line(Point(20,30), Point(180,165))
line.draw(win)
```

[oval]

```
oval = Oval(Point(20,150), Point(180,199))
oval.draw(win)
```

Chapter 4.4: Using Graphical objects

Class : describes the properties the instance will have
↳ every object is an instance of some class
↳ class examples: GraphWin, Point, Circle, etc.

Instances: different instances can vary in specific details!

[constructor] used to create a new instance of a class
↳ a special operation ^an expression* to create a brand new object

<class-name> (<param 1>, <param2>, ...)

the name of the class we want to create a new instance of

↳ parameters required to initialize the object. # and type of the parameters depends on the class

* often (but not always), a constructor is used on the right side of an assignment statement — the resulting object is immediately assigned to the the variable on the left (which is used to manipulate the object)

chapter 4.4 continued

the parameters (in the constructor) are stored as instance variables inside the object

To perform an operation on the object:

↳ methods: the messages sent to an object → functions living in the objects. invoked using dot notation

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

# and type of parameters: determined by the method used

↳ can be parameterless

accessor methods: allows us to access info from objects' instance variables

mutator methods: change the state of an object (change the values of its instance variables)

☆ Be careful of aliasing!

↳ when 2 different variables refer to exactly the same object; changes made through one variable reflects the other

↓

for graphics objects, to avoid this, use the clone method

# of pixels = resolution of the screen → determined by the monitor + graphics card in the computer

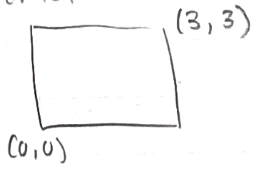
If we don't need to refer to an object again, we can simply create and draw the object immediately

ex) Text(Point(20,230), 'Hello').draw(win)

Can set coordinate system of graph window to be like cartesian plane:

win.setCoords(0.0, 0.0, 3.0, 3.0)

lower left upper right

↳ 

Class : 9/27/2021

Graphics : our first Python classes

graphics.py → library full of classes that will help us
- create graphics
- ↳ needs to be imported

- first step: create a graphics window where you can "draw"
  - graphics are NOT displayed in the shell like textual output
  - default graphics window: 200 × 200 pixels

alternative import statement

from graphics import * ← the more you do this, the slower the program gets

DON'T do it for the math library, even though it technically is still possible

on the graphics window:
- the origin (0,0) is at the top left
- x values increase from left to right
- y values increase from top to bottom
- in a 200 × 200 pixels, the bottom right corner is at (199,199) ← ☆ because 0 index

Objects: (referring to myGUI() function from class)

p1 = Point(40,80)

constructor

myFirstWindow = GraphWin() looks like no instance variables, but they are there we just don't see it.. like the default settings that it knows about itself

p2 = Point(130,100)

constructor

instance variables

↳ more "unseen" instance variables like color of point too...

Methods:

p1.draw(myFirstWindow)] an example of objects interacting with each other

action that the object is doing

p1 is being drawn on myFirstWindow

Class continued

Types of methods:

(Not all methods fit in these 2 categories)

[Accessors]

```
↳ >>> P1 = Point(80,40)
↳ >>> P1.getX() ☆
80
↳ >>> P1.getY() ☆
40
```

[Mutators]

```
↳ >>> p1.move(10,20) ☆
>>> p1.getX()
90
>>> p1.getY()
50
```

Chapter 4.7: Interactive Graphics

event-driven programming: draws interface elements (widgets) on the screen, then waits for user to do something

event → an object that encapsulates data about what just happened

↳ then it is sent off to be processed elsewhere in the program

[getMouse method] of the GraphWin class

↳ waits / pauses for the user to click somewhere on graphics window

↳ returns where the user clicks as a Point ← class

↳ can be used just to pause the program until the user has a chance to enter a value in ~~the~~ an input box

[getKey method] of the GraphWin class

↳ waits for the user to type a key on the keyboard,

↳ returns the user inputted letter as a string

Entry object:

↳ draws a box on the screen

↳ can contain text

↳ setText, getText methods just like Text object

↳ can be edited by the user

Class: 9/28/2021

GUI event driven programming: a style of coding where the flow is dictated by user input in the graphics window

↳ Basic structure:

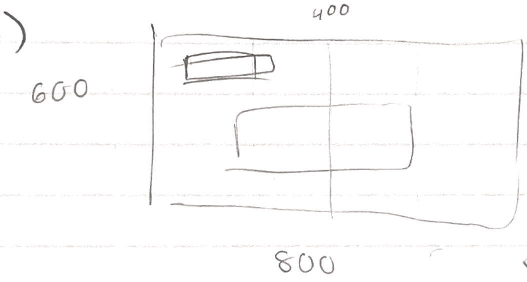
1. Draw interface element
2. Wait for user to input/interact → an event
3. Create an object that describes the event

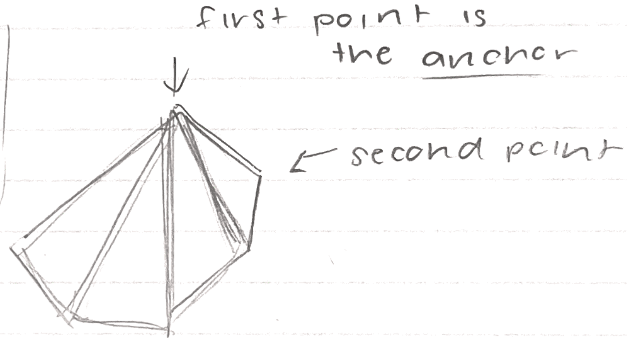
*still debug through the shell, even for graphics stuff

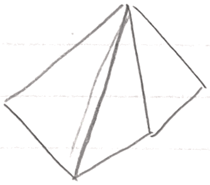
NOTE: can break up long print statements → just press return in the middle and it will line up.

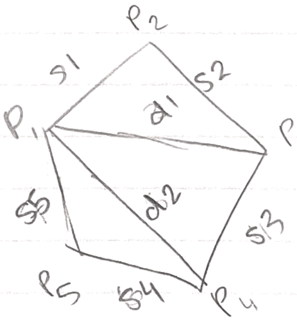
Lab: 04, graphics

2)









Δs1s2d1 + Δd1d2s3 + Δs4s5d2