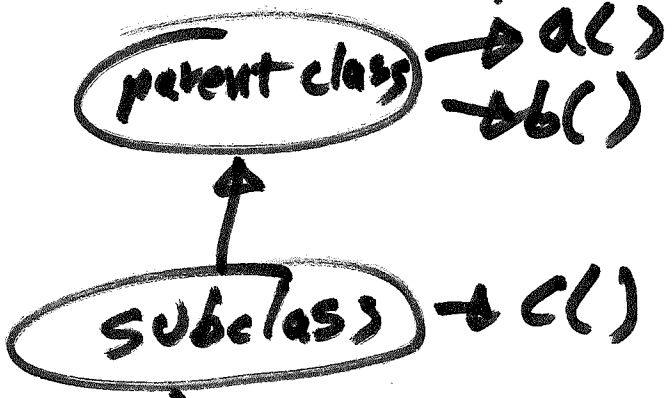
Object Oriented Programming

Inheritance

- A class definition may inherit from another class.
- This "subclass" will inherit the methods and variables of the "parent class"

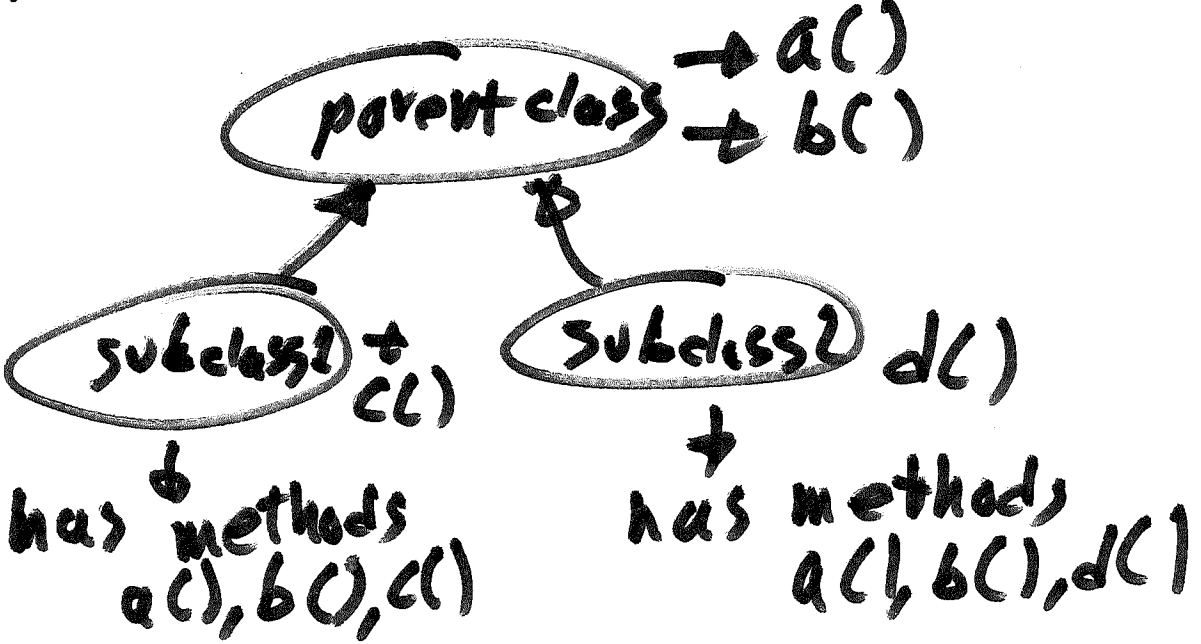
```
class Parent:
    def a():
        ⋮
    def b():
        ⋮
        ⋮
class Subclass(Parent):
        ⋮
```



subclass will have methods a(), b(), c()

subclass inherits from parent

- Also a parent class may have more than one subclass

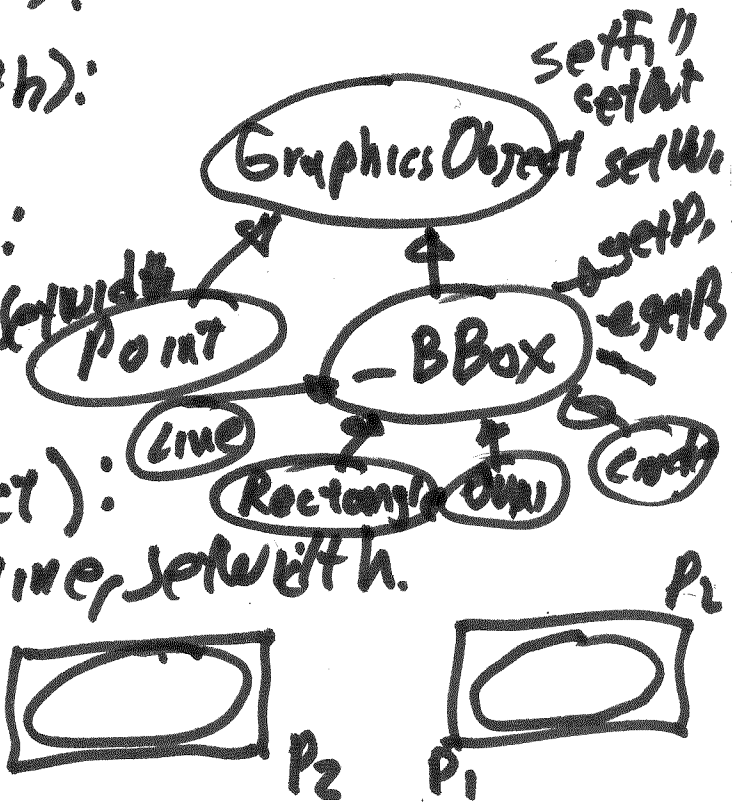


→ The subclass 1 and subclass 2 will inherit the methods in parent class so these methods do not need to be written twice.

In graphics.py

This is the graphics library provided by the author of the book

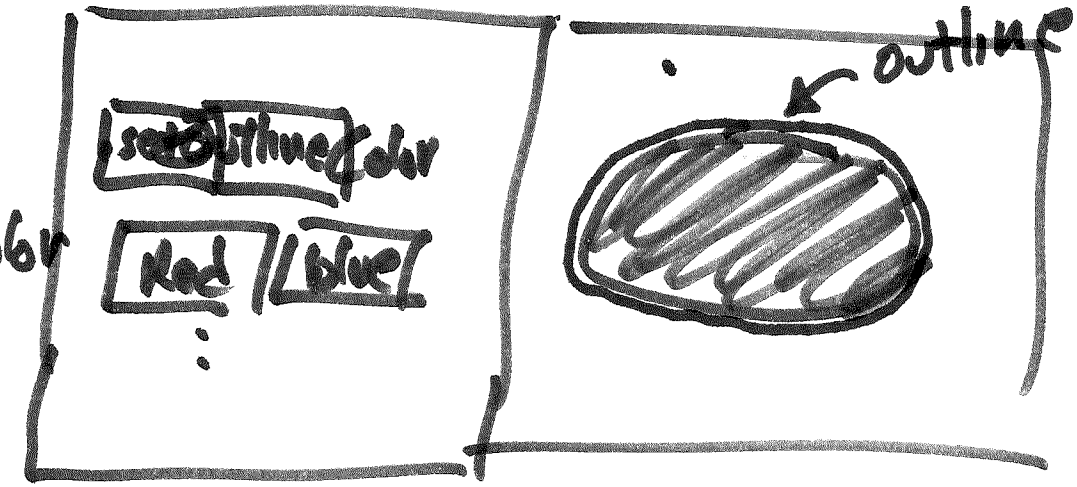
```
class GraphicsObject:
    def setFill(self, color):      # changes color
                                   # of a figure
    def setOutline(self, color):
    def setWidth(self, width):

class Point(GraphicsObject):
    # inherit setFill, setOutline, setWidth

class _BBox(GraphicsObject):
    # inherit setFill, setOutline, setWidth.
    def getP1(self)
    def getP2(self)
```



```
class Rectangle(_BBox)
        :
class Oval (_BBox)
class Circle(_BBox)
class Line (      )
class Polygon(    )
class Text (      )
```

- Example:
  In graphics editor add button to set the color of the outline of a figure.

To set outlinecolor
→ Click on color
→ Click on outLineColor button.



Steps editor.py

1. In createButtons add a new button.

```
self.addButton("OutlineColor", setOutlineColor)
```

2. In __init__ add

```
self.outlineColor = "black"
```

3. In newRectangle add

```
rect.setOutline(self.outlineColor)
```

Do same in newOval.

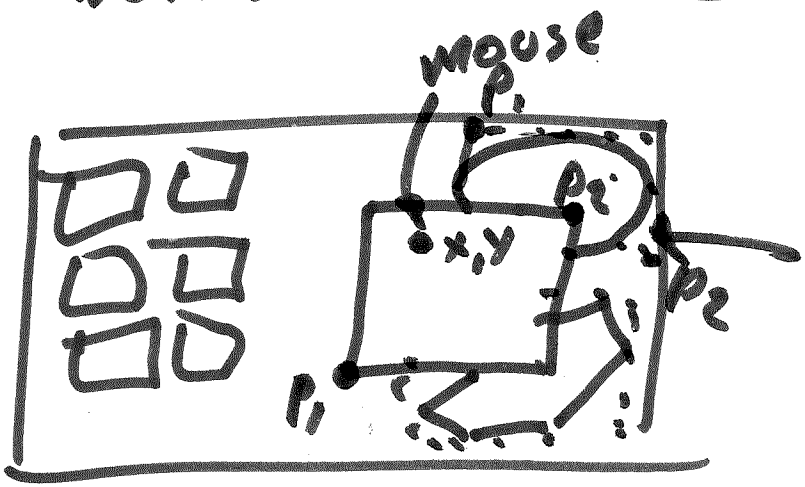
```
oval.setOutline...
```

4. Implement setOutlineColor used in button

```
def setOutlineColor(self, cmd):
    print("Set Outline to", self.color)
    self.outlineColor = self.color
```

Project 3

Finding a figure given a mouse coordinate (x, y)

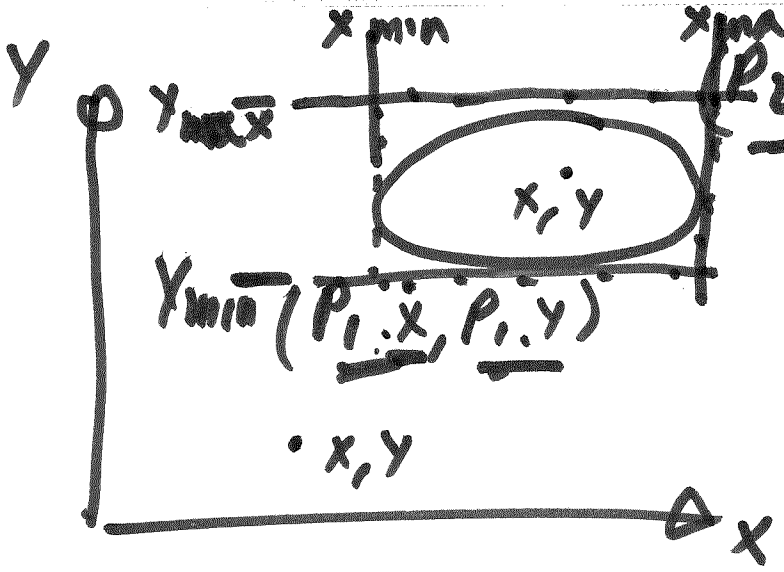
1. Press **Delete**
2. Print "With mouse select figure to delete"
3. 

Bounding Box

figure.P1 = O.getP1()
P2 = O.getP2()

P1 and P2 are the corner points that were selected to draw the figure.

4.

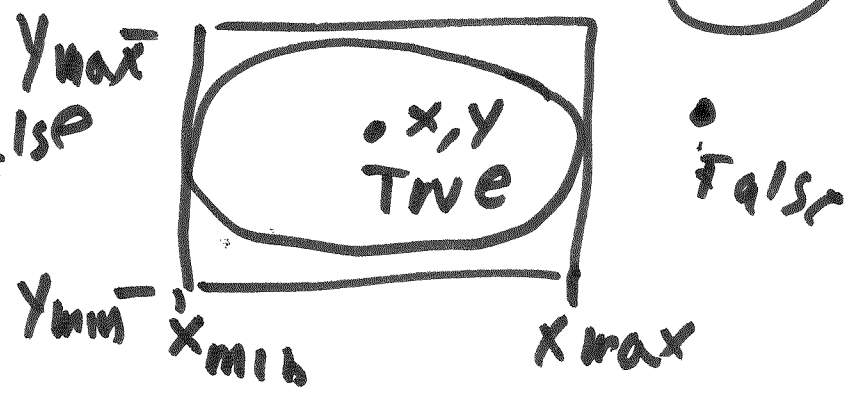


x, y is inside figure
if $x > p_1.x$ and $x < p_2.x$
and $y > p_1.y$ and $y < p_2.y$
Problem: $p_1.x$ may not be smaller than $p_2.x$ (same for y)

```
def isInsideFigure(self, fig, x, y):
    # Find xmin, xmax, ymin, ymax
    # from p1, p2
    xmin = p1.x
    p2.x
    xmin = p2.x
    xmax = p2.x
```

if $p_1.x < p_2.x$
$x_{min} = p_1.x$
$x_{max} = p_2.x$

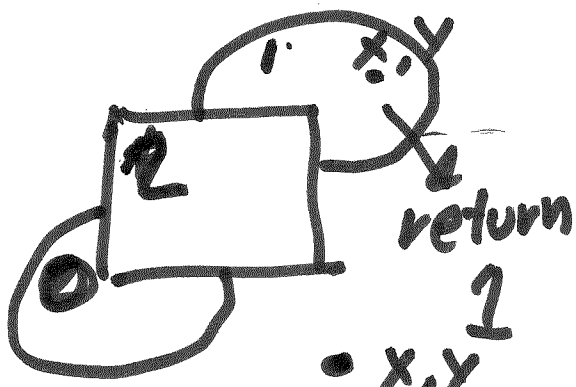
```
else
    xmin = p2.x
    xmax = p1.x
if p1.y < p2.y
    ymin = p1.y
    ymax = p2.y
else
    ymin = p2.y
    ymax = p1.y
if x > xmin and x < xmax and
   y > ymin and y < ymax:
    return True
else
    return False
```



```
def getSelectedFig(self, x, y):
    # Return index in list "figs"
    # that matches x, y
    # Returns -1 if no
    # figure found.
```

figs - List of all figures Increend

```
    i = len(figs) - 1
    while i >= 0:
        fig = figs[i]
        if self.isInsideFigure(fig, x, y):
            return i
        i = i - 1
    # We are here because no figure
    # was found. Return
    return -1
```



Algorithms

Program = Data Structure + Algorithm

Representation of the problem in the computer.
lists / dictionaries / variable etc.

Things you do with the data structure to solve the problem.

Sorting

Sorting is a typical algorithm that orders a list of numbers or strings in a certain order.

order numbers
ascending
descending

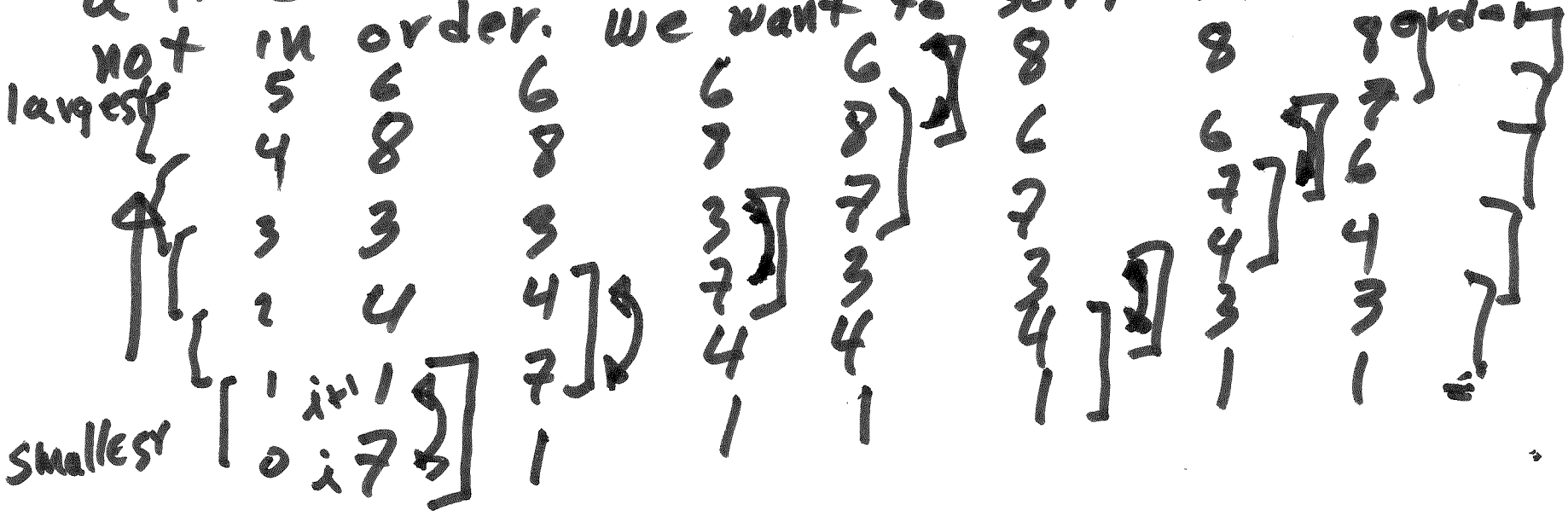
strings
alphabetical

Sorting Algorithms

bubble sort
quick sort
merge sort

BubbleSort

It compares a pair of items a pair at a time and it swaps them if the are not in order. we want to sort them in ascending order.



Implementation of bubblesort in Python

bubble.py

```
def sortlist(list):
    #Sort a list of numbers in "list" variable
    #Using bubblesort in ascending order
    swapNeeded=True
    while swapNeeded:
        swapNeeded=False
        for i in range(len(list)-1):
            if list[i] > list[i+1]:
                tmp = list[i]          #swap
                list[i]=list[i+1]
                list[i+1]=tmp
                swapNeeded=True
```

Searching

Problem: In a list find the position of a number given the number to search.

list [3, 4, 7, 1, 2, 9]
0 1 2 3 4 5

findPos(2, list) → 4 (4 is the index of element value 2)

Approaches

Linear Search:

Time = k n (n = length of list)
= O(n)

Compare element by element in the list until we find number.

```
def findPos(num, list):
    for i in range(len(list)):
        if list[i] == num:
            return i
    return -1
```

The time a linear search takes is proportional

Binary Search

This approach assumes that the list is sorted in ascending order.

Similar to the high/low game to guess a number.

list = [1, 3, 4, 5, 8, 9]
0 1 2 3 4 5

findPos(8, list): low=0 high=5

mid = (high+low)/2

mid = (5+0)/2 = 2

list[mid]=4 num

is 4 == 8 no

is 4 < 8 yes

low = mid + 1

low = 2+1 = 3

low=3 high=5

mid: $\frac{3+5}{2}=4$

list[mid]

list[4]=8

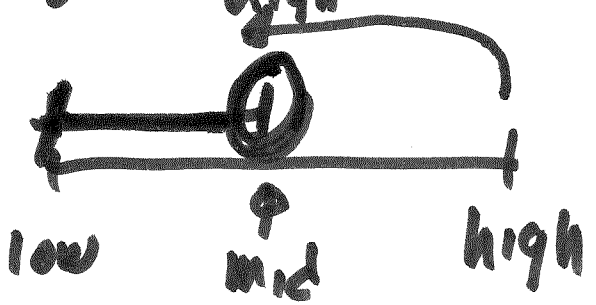
is 8==3 yes

return 4

Implementation of Binary Search

```
def findPos(num, lisy):
    #Assume list is sorted
    #in ascending order
    low = 0
    high = len(list) -1
    while high > low:    #iterate until high/low cross
                         #or we find num
        mid = (high+low)//2
        if list[mid] == num:
            #we found number
            return mid
        elif list[mid] > num
            high = mid-1
        else  #list[mid] < num
```

high<low



```
        low = mid + 1
    return -1
```

Binary Search is faster than linear search as long as the list is sorted.

Is Binary search is used only if there are many searches to be done in the same list since sorting takes time.

Time for binary search

$$\text{Time} = k \log(n)$$

$$= O(\log(n))$$

* size of list