Object oriented software engineering: Spatial Algorithms

Lecture and Workshop 2

Gary Watmough, Peter Alexander

Feedback on learning diary

Any problems from last week?

- The sorting solution code is now on learn: Week 1 > Coding Solutions

- Office hours:
  - Today 14:00 – 16:00
  - Wednesday 9:00 – 11:00
- Others: contact me by email gary.watmough@ed.ac.uk
- Office: G02 Drummond Old Library – Surgeons Square.

Week by week guide

1. Handling spatial data:
   a) Simple geometric calculations, distance and bearing, range searching and data sorting.
2. **Divide and Conquer**
   a) Binary searching, recursion and line generalisation
3. Grid data and arrays
   a) Handling, traversing and searching raster data. Point and focal functions.
4. Problem solving by task partitioning
   a) Nearest Neighbour Analysis and cartogram generation
5. Advanced raster and vector processing
   a) Developing flow routing algorithms, processing raw vector data

This week – intended learning outcomes

- be familiar with a range of algorithms used to manipulate and analyse spatial data
- develop python classes suited to the representation and analysis of spatial data

- Divide and Conquer Methods
- Binary Searching
- Recursion
- Line generalisation

Searching

Searching for a value

- Looking for a point in the data set
- Brute force or linear search
  - searches through every instance until success or failure.

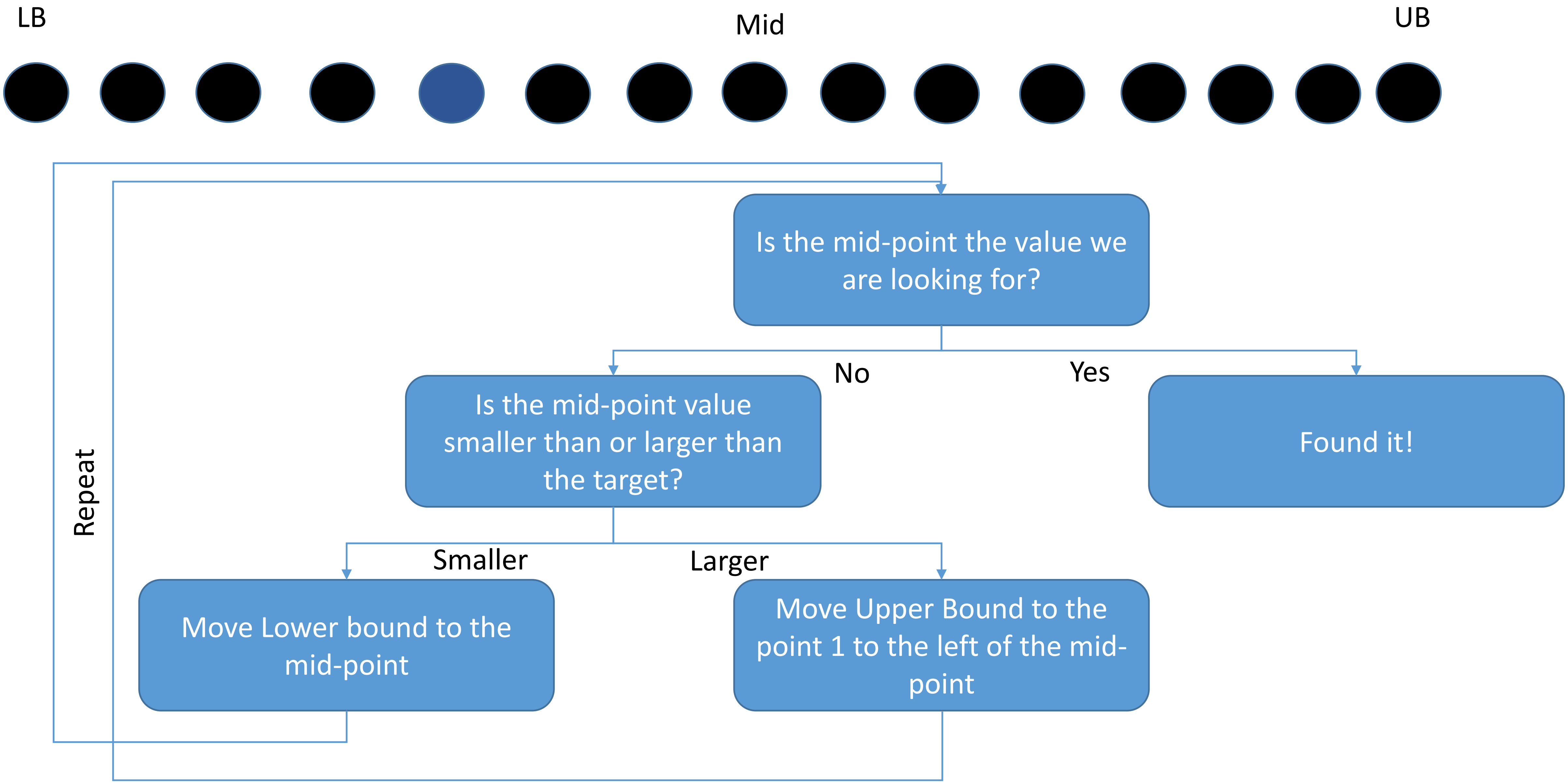
- Will find it, but will take time – becomes a problem in large lists
  - List length n results in search up to n times

Binary search

- Instead of looking at each item in list individually checks for an item in an array/list at the midpoint of that list.
- Then decides if it should search further up or further down the list

- Works with ordered lists

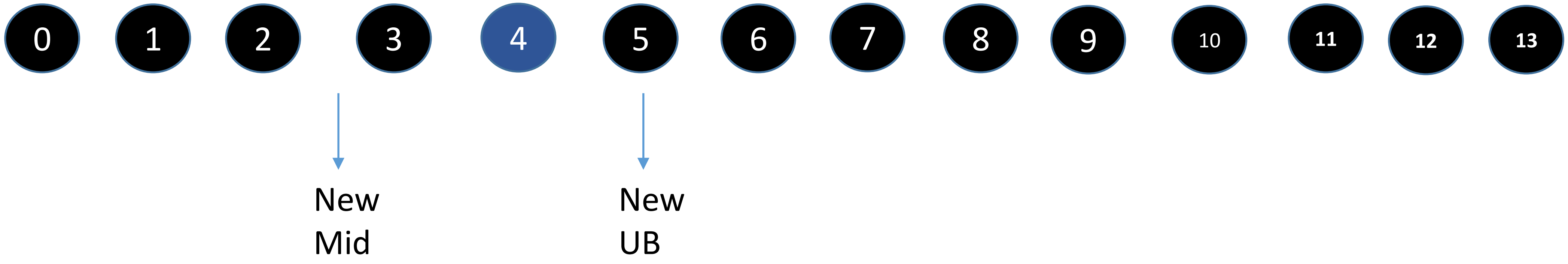
Binary search



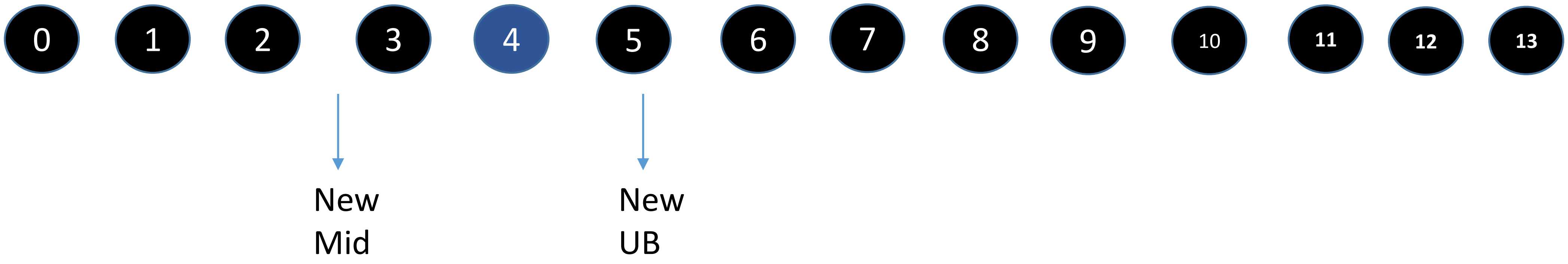
Binary search

Target = 4
Lower Bound = 0
Upper Bound = 13
Mid-point = 6

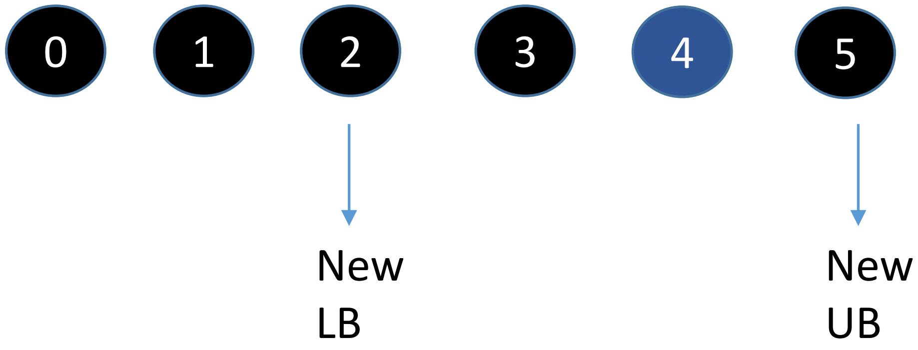
1. Is mid – point the value?
2. Is the mid point larger than or smaller than the target?
    1. Larger



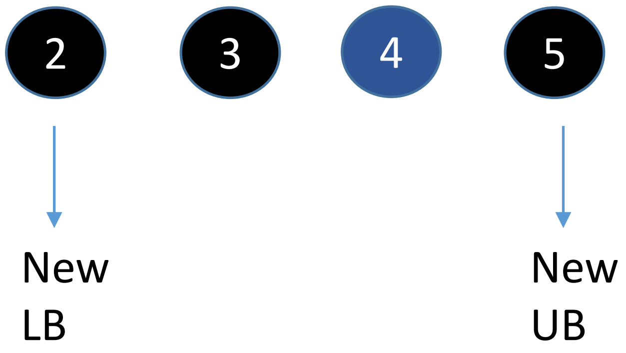
Binary search



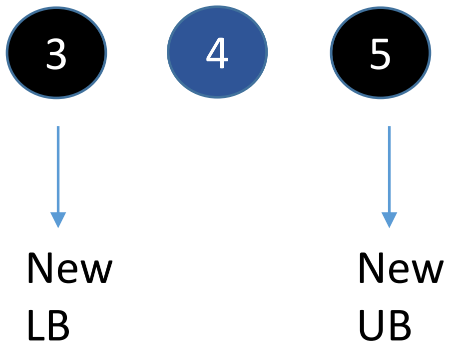
- Mid-point has to be integer (use floor division or rounding)
    - 2.5 = 2 (floor division)
- New Mid-point is lower than the target
- Lower Bound moves to mid-point



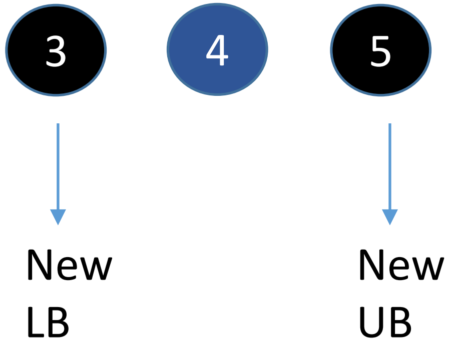
Binary search



- Mid-point has to be integer (use floor division or rounding)
  - 3.5 = 3 (floor division)
- New Mid-point is lower than the target
- Lower Bound moves to mid-point



Binary search



- Mid-point has to be integer (use floor division or rounding)
  - 4
- New Mid-point the value of the target

Coding problem 1: Binary Search

Task: Binary search – two examples

- What is the output showing you in each case?
    - Compare the two different binary search algorithms provided and identify the differences.

- Take some time to think these things through
    - When you think you know what's happening Comment the code that you have been given

- Change the print statements to make it clearer what the returned values represent

Binary Search 1:

Open:
Lecture_2A_Binary_searching

```
############################ Binary Search 1st Example ###

def binarySearch(alist, item):

    first = 0
    last = len(alist)-1
    found = False

    while first <=last and not found:
        midpoint = (first + last)//2
        if alist[midpoint]==item:
            found = True
        else:
            if item < alist[midpoint]:
                last = midpoint-1
            else:
                first = midpoint+1
    return found

testlist = [0,1,2,8,13,17,19,32,42]

```

Search for 13

Then search for 1

Then search for 3

Binary search 2

```
########################## Binary Search 2nd Example #######################

def binarySearch(s, mylist):
    lower = 0;
    upper = len(mylist)

    if len(mylist)==0:
        print('Nothing to search or does not have what we want to find.')
        return -1
    while (True):
        midpoint = (lower+upper)//2
        print (str(lower)+ " " + str(upper) + " " + str(midpoint))
        sm = mylist[midpoint]

        if (sm<s):
            lower = midpoint
        elif (sm>s):
            upper = midpoint
        else:
            return midpoint


testlist = [0,1,2,8,13,17,19,32,42]

print(binarySearch(13, testlist))
```

Use the same test list from last slide:

Search for 13

Then search for 1

Then search for 3

Advanced Task: Binary Search

- In the second example: what is happening when we search for a value that isn't in the list?
- Why is this happening?
- Can we fix this?

- See if you can add in a statement that stops the algorithm from entering an infinite loop

You need to add a statement in the code:

Binary search take home message

- Using the mid-point we either find the target or we split the list of targets in half.

- It is a divide and conquer approach:
  - we divide the problem into smaller pieces,
  - solve the smaller pieces in some way, and
  - then reassemble the whole problem to get the result.

Any Questions?

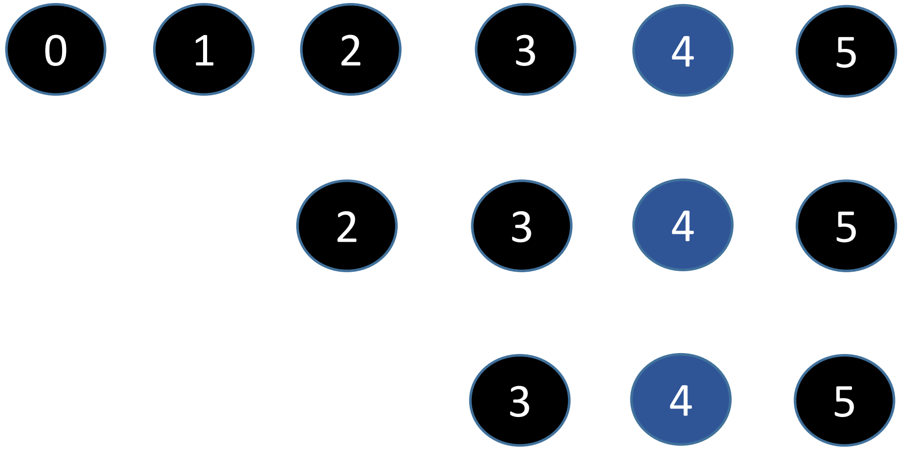
Break time

Recursion

- Instead of solving a hard problem
- Turn it into a slightly easier version of the same problem
- Recursion – is when a function calls itself

- In this search example we are basically doing the same operation repeatedly.
- We search part of a list, and then ‘split’ the list, searching only the top or bottom part.
- This is an ideal candidate for a common programming technique known as *recursion*.

Recursion



We can write a method, whose arguments are:

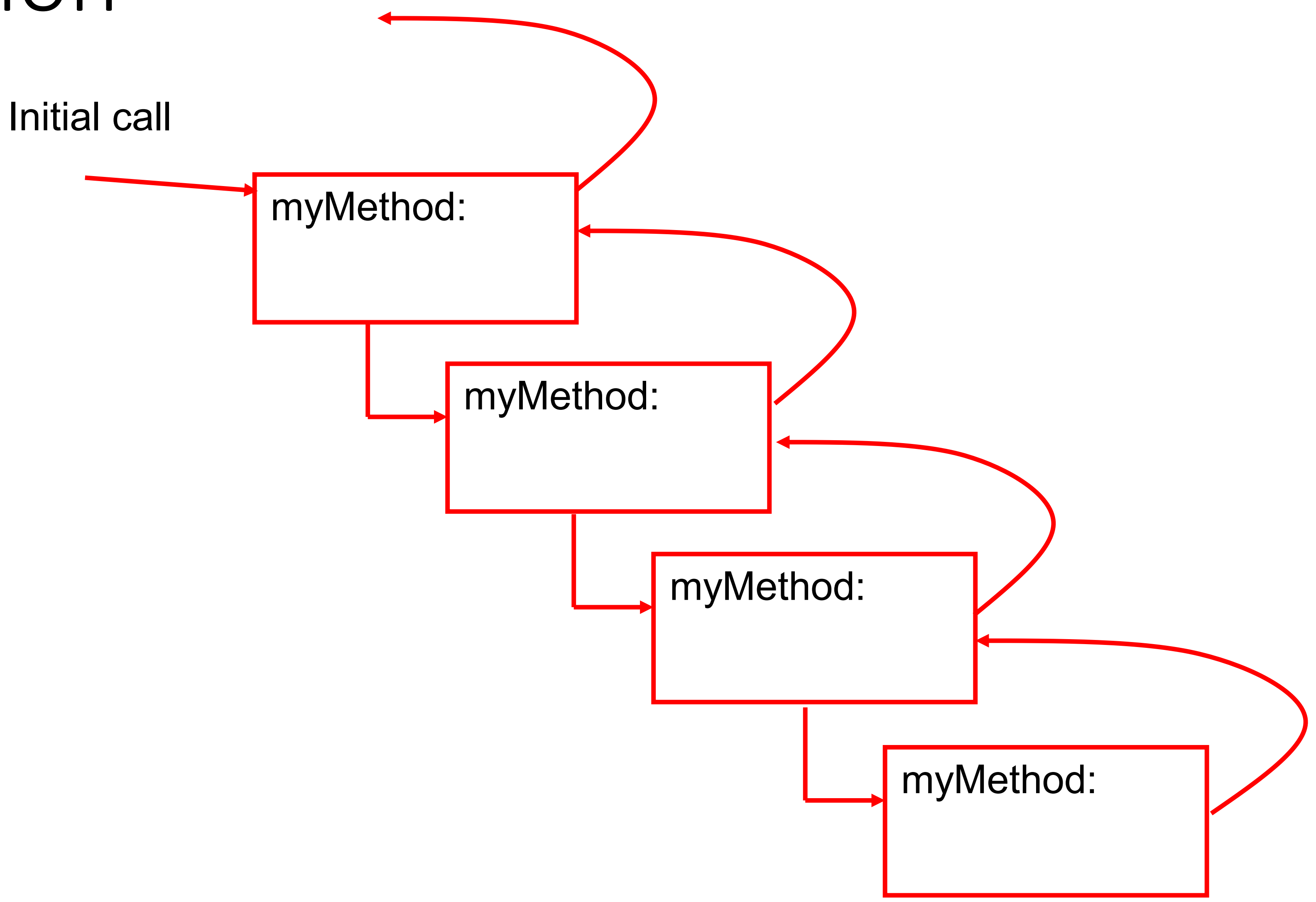
what we want to find

what we want to search;

and that then returns the location if found.

If we don’t find things first time, we can simply call the routine again with a smaller subset of the problem and keep going until we do.

Recursion



Coding problem 2:
Binary Search with recursion

Coding problem 2: recursion

```
def bSearch(key, mylist, left, right):
    if (left >= right):
        return -1
    mid = (left+right)//2
    sm = mylist[mid]
    print ('search now centred at:{} '.format(sm))

    if (sm < key):
        print ("  moving search to right")
        return bSearch(key, mylist, mid+1, right)
    elif (sm > key):
        print ("  moving search to left")
        return bSearch(key, mylist, left, mid)
    else:
        return mid
```

Open:
Lecture 2_B Recursion

Coding problem 2: recursion

```
def bSearch(key, mylist, left, right):
    if (left >= right):
        return -1
    mid = (left+right)//2
    sm = mylist[mid]
    print ('search now centred at:{} '.format(sm))

    if (sm < key):
        print ("  moving search to right")
        return bSearch(key, mylist, mid+1, right)
    elif (sm > key):
        print ("  moving search to left")
        return bSearch(key, mylist, left, mid)
    else:
        return mid
```

These are the important bits. This is a self-reference

Coding problem 2: recursion

- Recursion Code - provided
- Add comments to your script to explain what is going on.
- We will discuss this together before we move on.

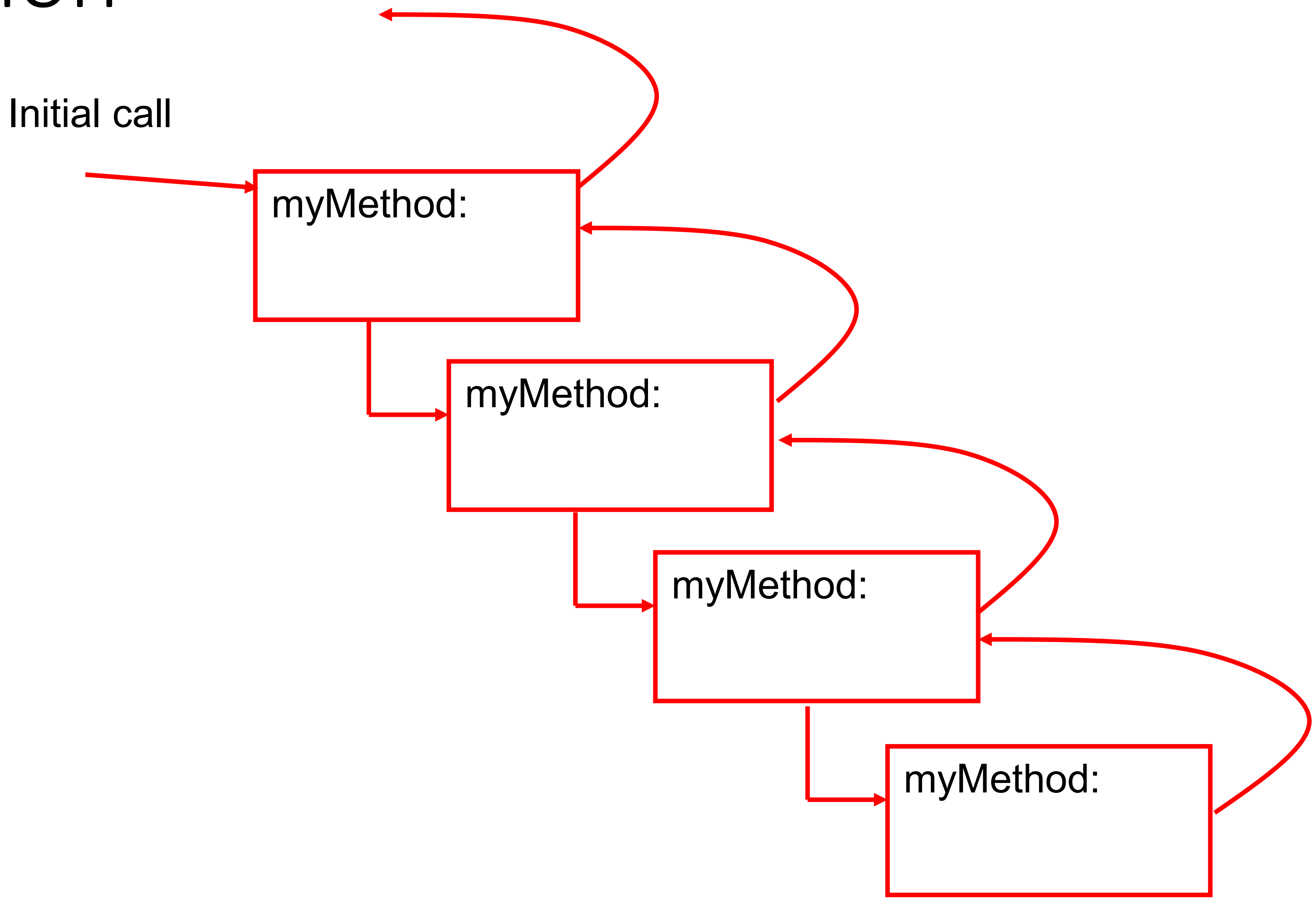
Coding problem 2: recursion

```python
def bSearch(s, mylist, left, right):
    if (left>= right):
        return -1

    m=(left+right)//2

    sm=mylist[m]
    print ("Search now centred at {}, position {}".forma

    if (sm<s):
        print (" * moving search to right")
        return bSearch(s, mylist, m+1, right)
    elif (sm>s):
        print (" * moving search to left")
        return bSearch(s, mylist, left, m+1)
    else:
        return m
```

Call again with subset to right

Call again with subset to left

Recursion



Coding problem 3: Find ID in list

- Define function to generate random IDs, these should be:
  - 6 characters long
  - Be a mix of uppercase letters and numbers

```python

def id_generator(size=6, chars=string.ascii_uppercase + string.digits):
    return ''.join(random.choice(chars) for x in range(size))

```

- You can add this to the recursion module you just loaded.

Coding problem 3: Find ID in list

- Create a list of 100 IDs using the id_generator defined above

```python

longl=[]
for p in range(num):
    longl.append(id_generator())
    print (longl[p])

n=random.randint(0, num)

```

Coding problem 3: Find ID in list

- Randomly identify a single ID number from our list above and use the binary search to find it.

```
n=random.randint(0,num)

print "search for" + longl[n]

found=binarySearch(longl[n],longl)
if (found==-1):
    print "String not found"
else:
    print "string found at position:" + str(found)
```

Coding problem 3 result

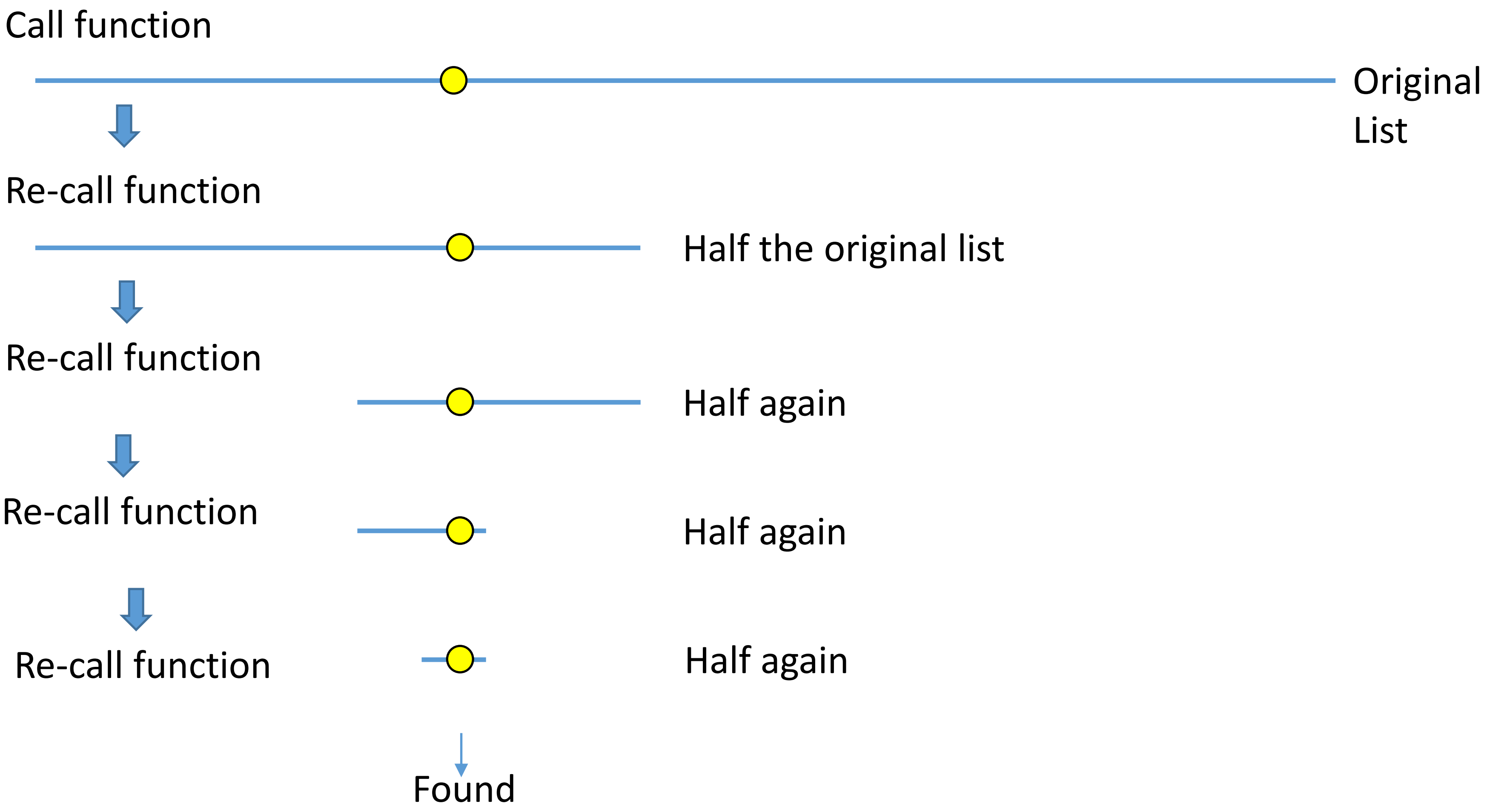
```
'6A7OFX', '3NKUHX', 'MCVWSH', 'RPNT9T', 'DOLB37', 'JP0F5L', 'S9V9G1', 'H9GBZN', 'HUNT3X', 'MDNJDL']
['06EP5X', '1IC8W2', '1PQ26F', '22IX9A', '2N8Q5N', '2TYGNJ', '31WXF6', '38T3XG', '3KMBW6', '3N5QJK', '3NKUHX', '4D3Y62', '5CPW77', '5NN2FS', '5RFREF',
'6A7OFX', '6IIFUL', '6IQGVD', '6QS3O8', '73GBFZ', '7B3JIM', '87C374', '8EHZIH', '8HOAXW', '8PLIHE', '9PUWX0', '9TX7NN', '9Z8JOD', 'AKQIBX', 'ARP4UE',
'B87ITA', 'BR3V1Q', 'C37IPR', 'CAMHOK', 'CFU5FO', 'CMY6BV', 'DOLB37', 'DYN2XK', 'E1ZG57', 'E4FXN1', 'EKRTTC', 'FGU7LZ', 'FZQMMX', 'H9GBZN', 'HEU7RQ',
'HUNT3X', 'I4WZU6', 'IWM3RM', 'JFW6UO', 'JFWC4N', 'JP0F5L', 'K1YR5Y', 'K8GHMQ', 'KCO8A2', 'L5H704', 'LJBKYN', 'LWNUG8', 'M10ZY9', 'MADT1X', 'MBF3KP',
'MCVWSH', 'MDNJDL', 'MUID0D', 'N0613R', 'N0E3KP', 'NNREM0', 'O2V8ZK', 'OAI92K', 'PSZ2B4', 'PZQKU2', 'RECBVM', 'RPNT9T', 'RVKHQH', 'S45N7C', 'S4MDU8',
'S925TS', 'S9V9G1', 'SVIZZE', 'TAWZDK', 'TU2G17', 'U2AIQT', 'U4NQHZ', 'UA8ABP', 'UZPIR0', 'VEP6KN', 'VGPI8F', 'VU2INN', 'W64UES', 'WCE8O0', 'WO5TT3',
'X68JM1', 'X9ECCF', 'XOS0BG', 'XUTCHU', 'YRCR81', 'Z1P0BB', 'ZA6080', 'ZLCAXK', 'ZN87CG', 'ZPWQSB']
Searching for CMY6BV
Search now centred at JP0F5L, position 50
 * moving search to left
Search now centred at 9PUWX0, position 25
 * moving search to right
Search now centred at E1ZG57, position 38
 * moving search to left
Search now centred at C37IPR, position 32
 * moving search to right
Search now centred at DOLB37, position 36
 * moving search to left
Search now centred at CMY6BV, position 35
Found CMY6BV at position 35

In [13]:
```

Does it work for everyone?
Try running the code a few times with different list sizes

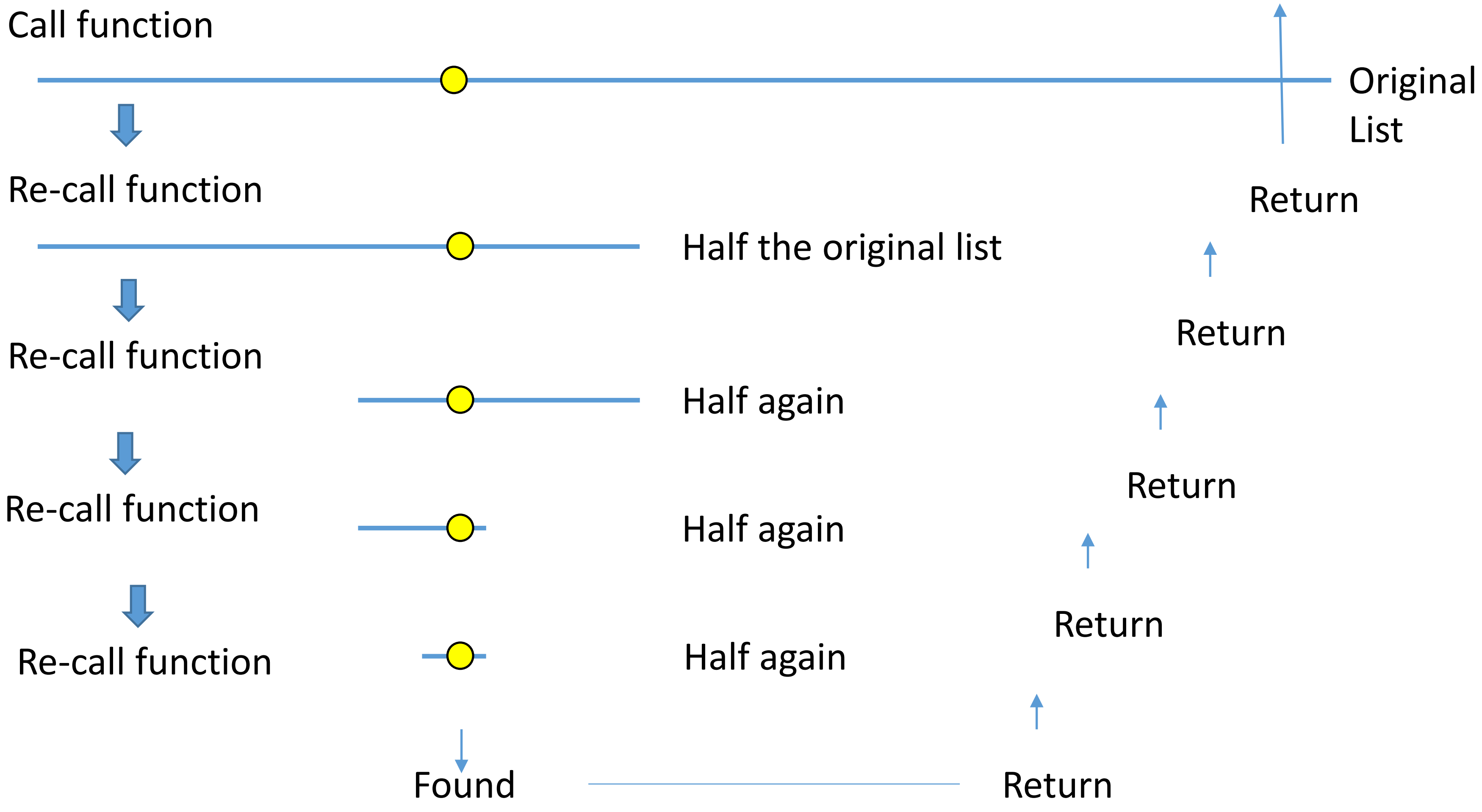
Without recursion

- Returning to our original binary search algorithm
- How does this work?

Another way to do this search?



Another way to do this search?



In this case the call looks like:

def bSearch(s, mylist)

- Each time you only need pass the list you’re searching which gets increasingly small

- Could you modify your recursive search to do things this way

- Is there any great change of search speed?

- What might be the problem in terms of what we really want the function to do?

- Can you think of some ways round this?

Different recursion method: Solution slide 3

Console 1/A

```
'MCJVE6', 'MCRFWJ', 'MCTPAY', 'MCXB7Y', 'MD769Q', 'MD7VDU', 'MDEVGB', 'MDJKZ5', 'MDKT0V', 'MDOEFA', 'MDQLE0', 'MDRE9K', 'MDRH9R',
'MDSW19', 'ME50J2', 'MEEEV4', 'MEKZH5', 'MENB1J', 'MEPMIN', 'MEY043', 'MEYR6X', 'MF75LH', 'MF892U', 'MFMFKD', 'MFMT1W', 'MG1M5E',
'MG7EXD', 'MG9V2X', 'MGC40Y', 'MGCD4E', 'MGOR8N', 'MGQUN0', 'MGZAD1', 'MH2J93', 'MH2UPB', 'MH687Z', 'MHC79Q', 'MHIJ5C']
Search now centred at M7Q959, position 77
 * moving search to left
['LX796Z', 'LX8SHL', 'LX9L52', 'LXAO4H', 'LXOTR6', 'LXQWMR', 'LXXQKV', 'LXZTTA', 'LY392R', 'LY4AVI', 'LY7W5X', 'LYCPZV', 'LYH10A',
'LYO9Y6', 'LZ93XV', 'LZ9ATR', 'LZANSU', 'LZB57M', 'LZGZD1', 'LZJY9P', 'LZKO13', 'LZRXPZ', 'LZWHB0', 'LZZGV8', 'LZZVUL', 'M03G5Z',
'M0458X', 'M06FDE', 'M089YH', 'M0D33J', 'M0L62N', 'M0WUY2', 'M11WR7', 'M1C11M', 'M1CR18', 'M1PG13', 'M1VYF5', 'M2JAJ9', 'M2KUPJ',
'M2WET9', 'M3241V', 'M34VKU', 'M3DM52', 'M3EDXH', 'M3F0J1', 'M3GKN6', 'M3KF18', 'M3RQJ3', 'M3U18M', 'M3UH8E', 'M41UDA', 'M4BB1B',
'M4KCSC', 'M4MXB8', 'M4PVJK', 'M4ROMO', 'M4WEXA', 'M4XTU0', 'M5BV21', 'M5F5WX', 'M5GZTF', 'M5XDMF', 'M66H5A', 'M66OCN', 'M6G1OD',
'M6HBJ0', 'M6KTHI', 'M6WRDK', 'M6XHVR', 'M72EIN', 'M72OAA', 'M76SVG', 'M78A02', 'M79HJ6', 'M7C48P', 'M7EUNU', 'M7JJEY']
Search now centred at M2KUPJ, position 38
 * moving search to right
['M2WET9', 'M3241V', 'M34VKU', 'M3DM52', 'M3EDXH', 'M3F0J1', 'M3GKN6', 'M3KF18', 'M3RQJ3', 'M3U18M', 'M3UH8E', 'M41UDA', 'M4BB1B',
'M4KCSC', 'M4MXB8', 'M4PVJK', 'M4ROMO', 'M4WEXA', 'M4XTU0', 'M5BV21', 'M5F5WX', 'M5GZTF', 'M5XDMF', 'M66H5A', 'M66OCN', 'M6G1OD',
'M6HBJ0', 'M6KTHI', 'M6WRDK', 'M6XHVR', 'M72EIN', 'M72OAA', 'M76SVG', 'M78A02', 'M79HJ6', 'M7C48P', 'M7EUNU', 'M7JJEY']
Search now centred at M5BV21, position 19
 * moving search to left
['M2WET9', 'M3241V', 'M34VKU', 'M3DM52', 'M3EDXH', 'M3F0J1', 'M3GKN6', 'M3KF18', 'M3RQJ3', 'M3U18M', 'M3UH8E', 'M41UDA', 'M4BB1B',
'M4KCSC', 'M4MXB8', 'M4PVJK', 'M4ROMO', 'M4WEXA', 'M4XTU0']
Search now centred at M3U18M, position 9
 * moving search to right
['M3UH8E', 'M41UDA', 'M4BB1B', 'M4KCSC', 'M4MXB8', 'M4PVJK', 'M4ROMO', 'M4WEXA', 'M4XTU0']
Search now centred at M4MXB8, position 4
 * moving search to right
['M4PVJK', 'M4ROMO', 'M4WEXA', 'M4XTU0']
Search now centred at M4WEXA, position 2
Found M4WEXA at position 2
M4WEXA
```

Recursion – summary

- You can implement recursion for all sorts of tasks where you repeat operations on a different subset of data to focus in on a solution.
- Typically to use recursion you need to make sure you to specify what the data subset is.
- You need to ensure that you return a value that is passed (recursively) back to the previous method call but which can provide your ‘answer’ from your first call.

Any Questions?

Take a break

Objects, classes, hierarchy, inheritance

Review of important OOP principles before we move into spatial analysis

This should be revision from previous semester, if you are still struggling with this you need to take some time this week to work on it in your independent learning

Review of OPP classes

- Questions for the group:
  - What is an object?
  - What is object oriented programming?
  - What is a class?

Classes and Objects

- Objects:
  - Ways of organising code to make complex ideas easier to think about
- In Python we can define objects using classes
- Define a class using class keyword

```python
class giraffe:
    def __init__(self, spots):
        self.giraffe_spots=spots

frank = giraffe(100)
print(frank.giraffe_spots)
```

Class

```
class FirstClass:
    def setdata(self, value): # define class method
        self.data = value # self is the instance
    def display(self):
        print (self.data) #self.data: per instance
# this means that we assign the names setdata and display in the class statement
# so it generates attributes attached to the class:
# FirstClass.setdata and FirstClass.display

## create two new instances
x = FirstClass()
y = FirstClass() # each of these is a new namespace

# these namespaces have access to their classes attributes
# in OOP we have three objects, two instances and 1 class.

x.setdata("King Arthur")
y.setdata(3.14159)

x.display()
y.display()

#change instance attributes either in the class itself by assigning to self
# or  outside the class by assigning to an explicit instance object,
x.data = "New value"
x.display()
```

Self?

- Everyone ok with the self variable that keeps appearing?

- What is it?

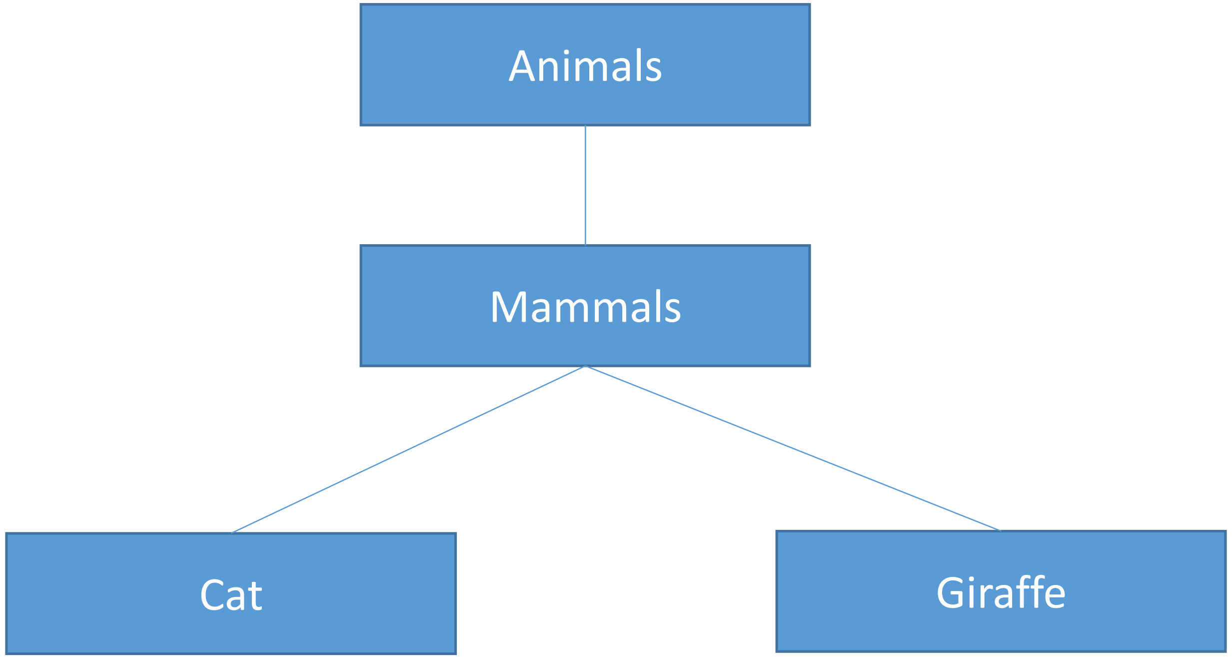
Class hierarchy and inheritance

Animals is the broadest class

It is the parent of mammals

Which is the parent of cat and giraffe

Giraffe and Cat are children of Mammals and Animals



Class and hierarchy

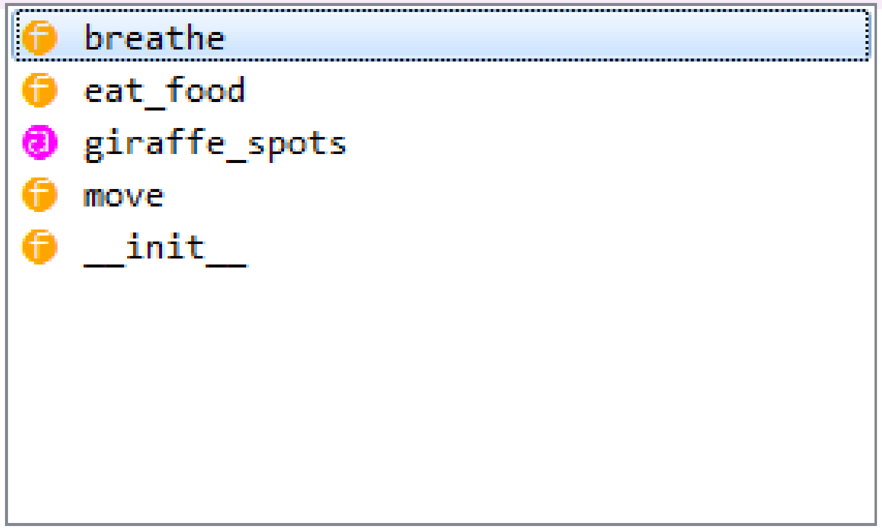
In this example:

Animals class is the parent of giraffe

If we create a giraffe object called frank with 100 spots

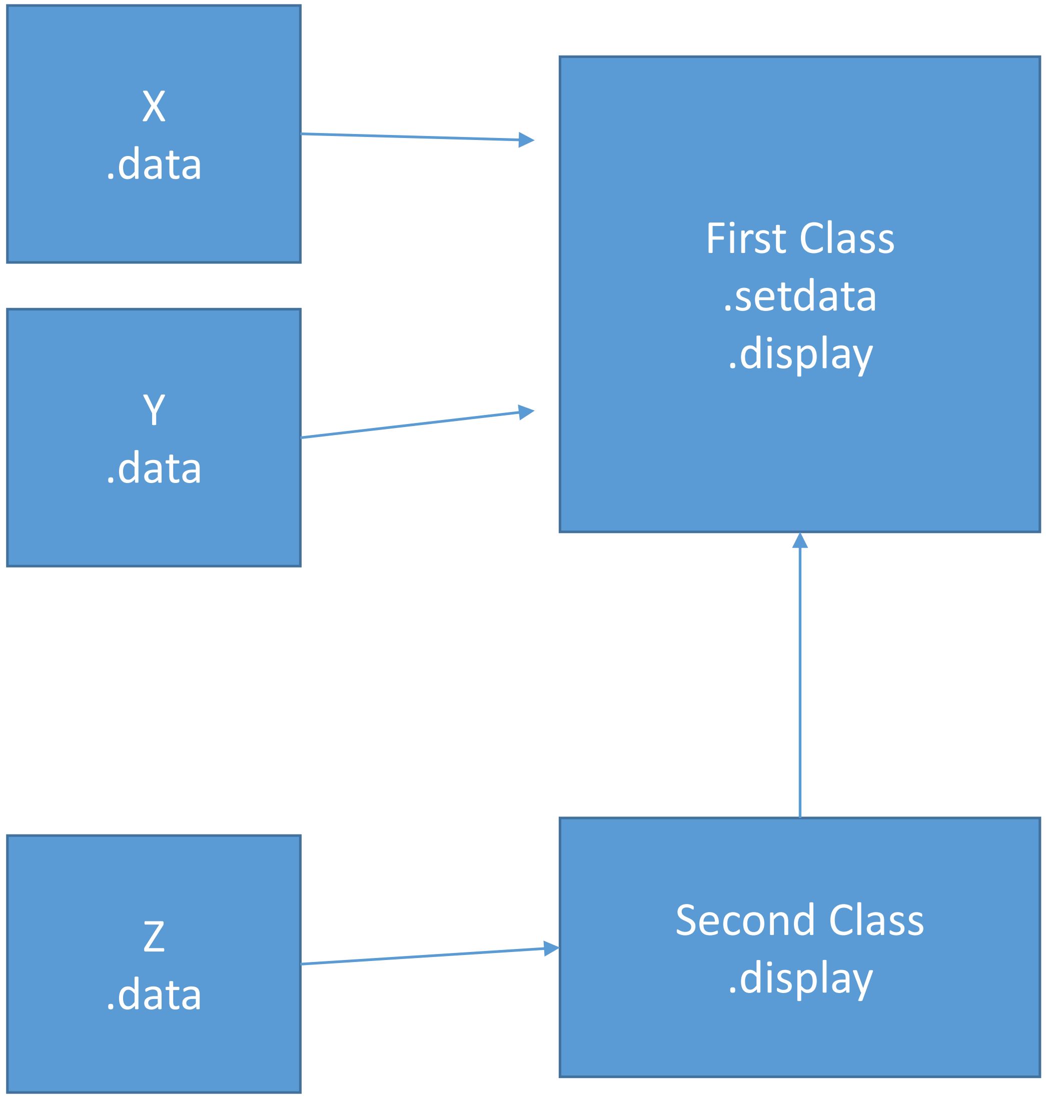
Frank can take other arguments that were defined in animals – this is inheritance

```
class animals:
    def breathe(self):
        pass
    def move(self):
        pass
    def eat_food(self):
        pass

class giraffe(animals):
    def __init__(self, spots):
        self.giraffe_spots=spots

frank = giraffe(100)

frank.
```



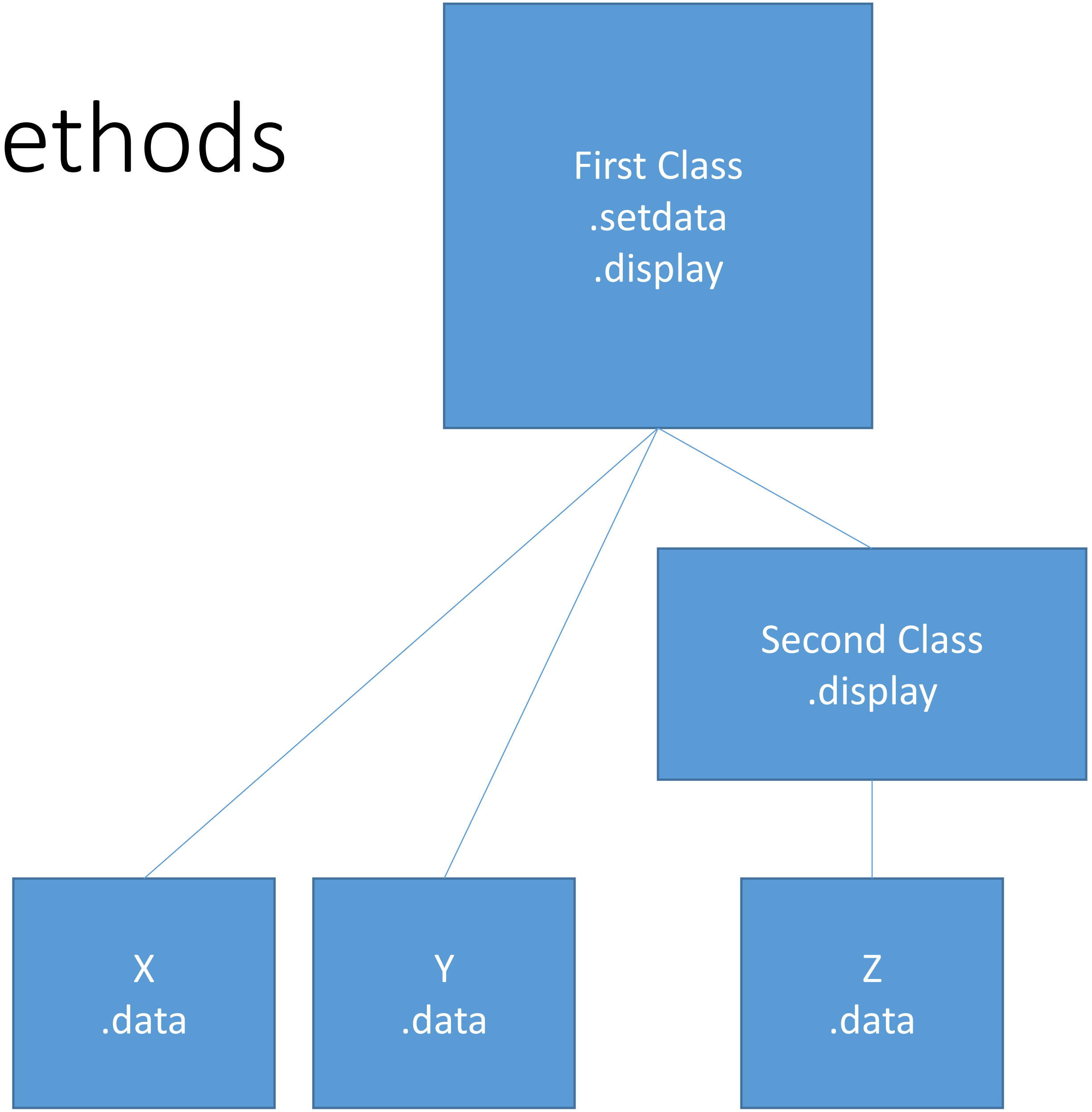
Inheritance

- Superclass listed in brackets of a class header
- Classes inherit attributes from superclasses
- Instances inherit attributes from all accessible classes
    - Class they are generated from
    - All superclasses
- Each object.attribute reference invokes a new search
    - Bottom up
    - Left to right



Specialising inherited methods

- Because of the way that search works:
- Replacing attributes by redefining the lower in the tree is possible
  - Z takes the display attributes from second class changes
  - X and Y take the display attributes from FirstClass



Class hierarchy and inheritance

```python
class FirstClass:
    def setdata(self, value):
        self.data = value
    def display(self):
        print(self.data)

class SecondClass(FirstClass):
    def display(self):
        print('Current value = %s' % self.data)

x=FirstClass()
y=FirstClass()
x.setdata("King Arthur")
y.setdata(3.14159)
z=SecondClass()
z.setdata(99)

x.display()
y.display()
z.display()
```

Example from last week

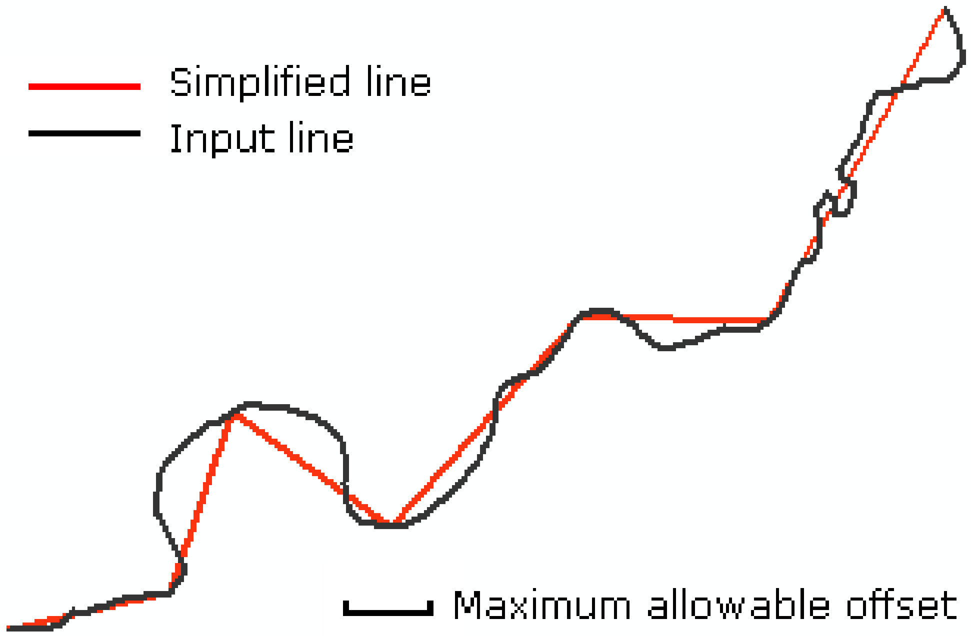
```
"""

import math

class Point2D(object):
    '''A class to represent 2-D points'''

# The initialisation methods used to instantiate an instance
    def __init__(self,x,y):
#ensure points are always reals
        self._x=x*1.
        self._y=y*1.

#return a clone of self (another identical Point object)
    def clone(self):
        return Point2D(self._x,self._y)

#return x coordinate
    def get_x(self):
        return self._x

#return y coordinate
    def get_y(self):
        return self._y

```

- In module Points.py

Summary

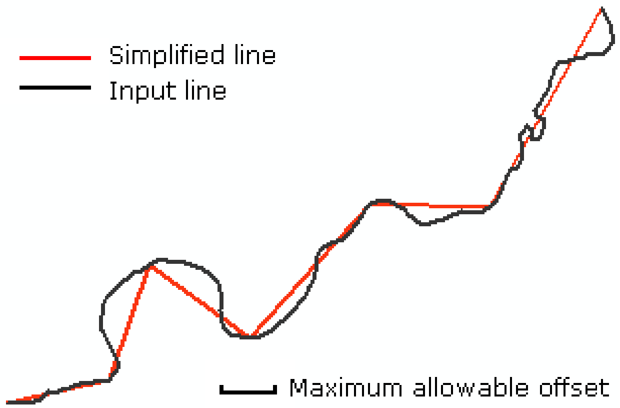
- Classes – define what things are in Python
- Instances – inherit the class attributes
- Classes can inherit from other classes
  - Allowing for hierarchies of classes
- 3D points inherit 2D point attributes and add a third dimension
- Means we don’t need to repeat our code when defining 3D points.

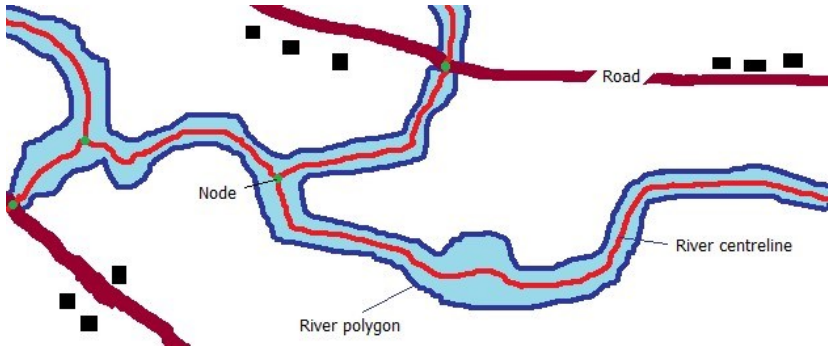
Spatial analysis: Line Generalisation



Line generalisation

- Simplification – eliminates detail
- Collapsing – reduction of line or area features to point features or area features to line features. For example, river polygon collapsed to a single line representing the middle.

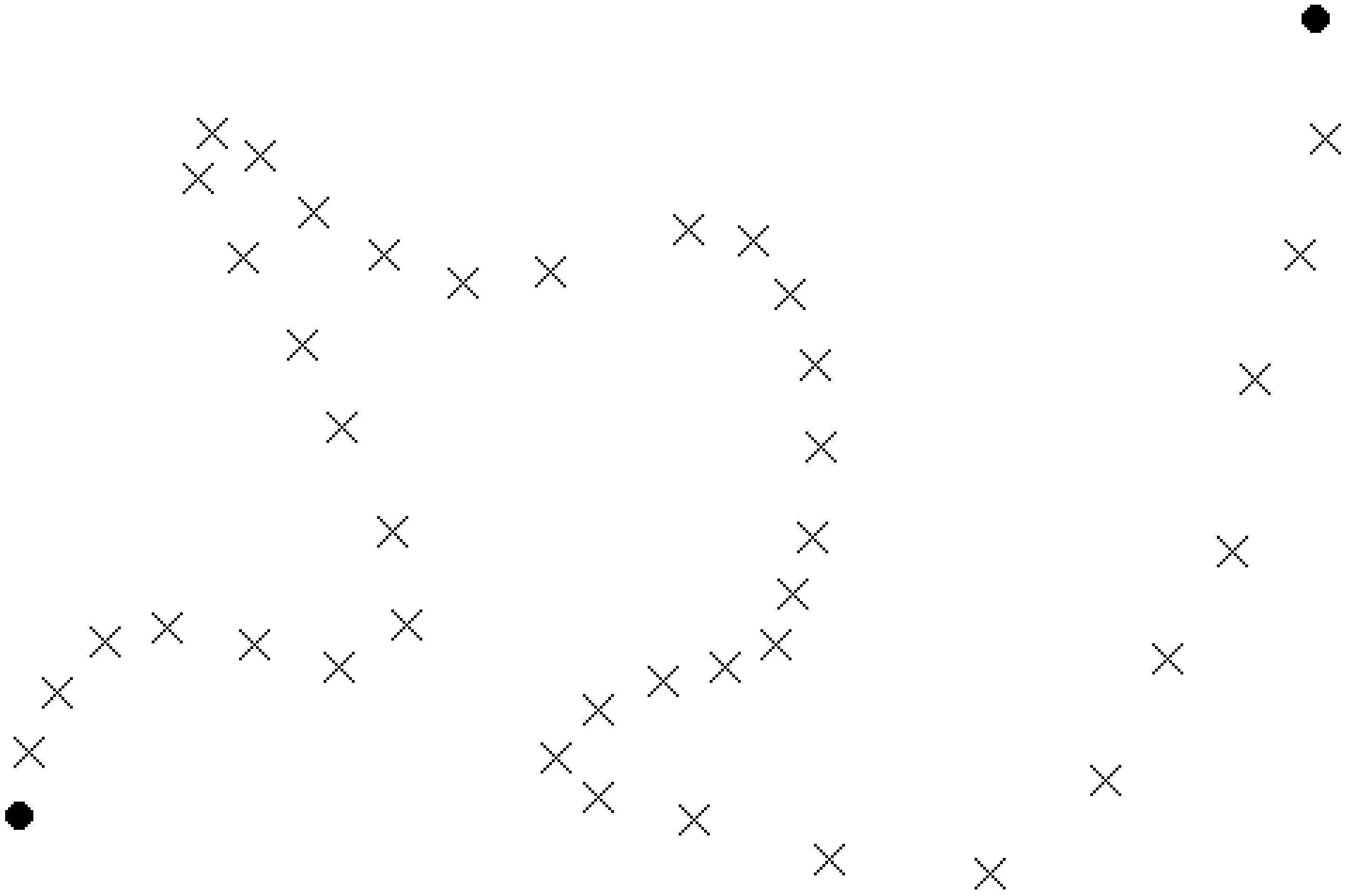




Douglas-Peuker Line Generalisation

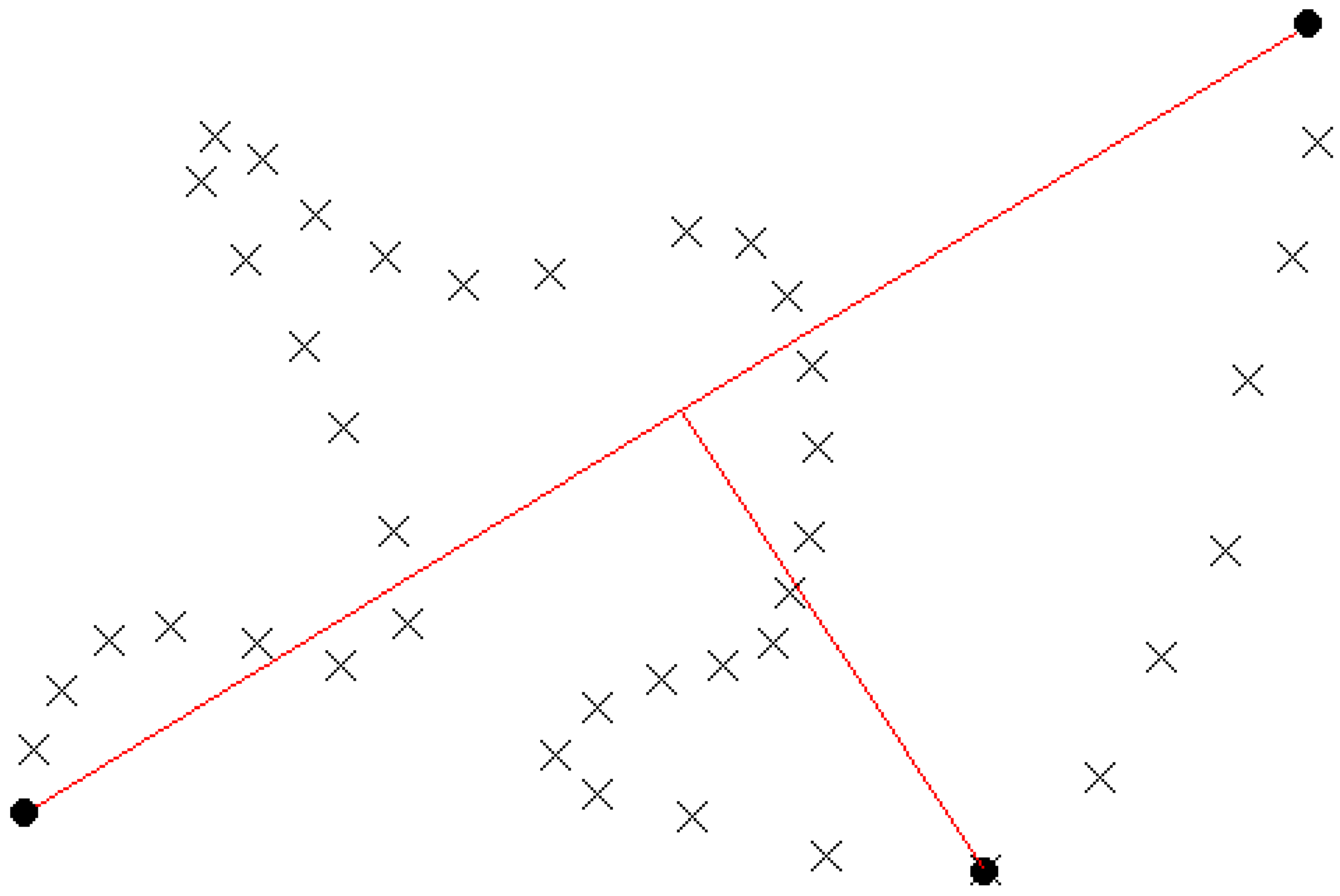
- Simplification – eliminates detail
- In GIS systems it is often desirable to remove unnecessary vertices. These may have been generated by over-sampling during digitisation.
- The Douglas-Peuker line generalisation algorithm works by reducing a point set by removal of vertices if they fall within a bandwidth tolerance.

Douglas-Peuker Line Generalisation



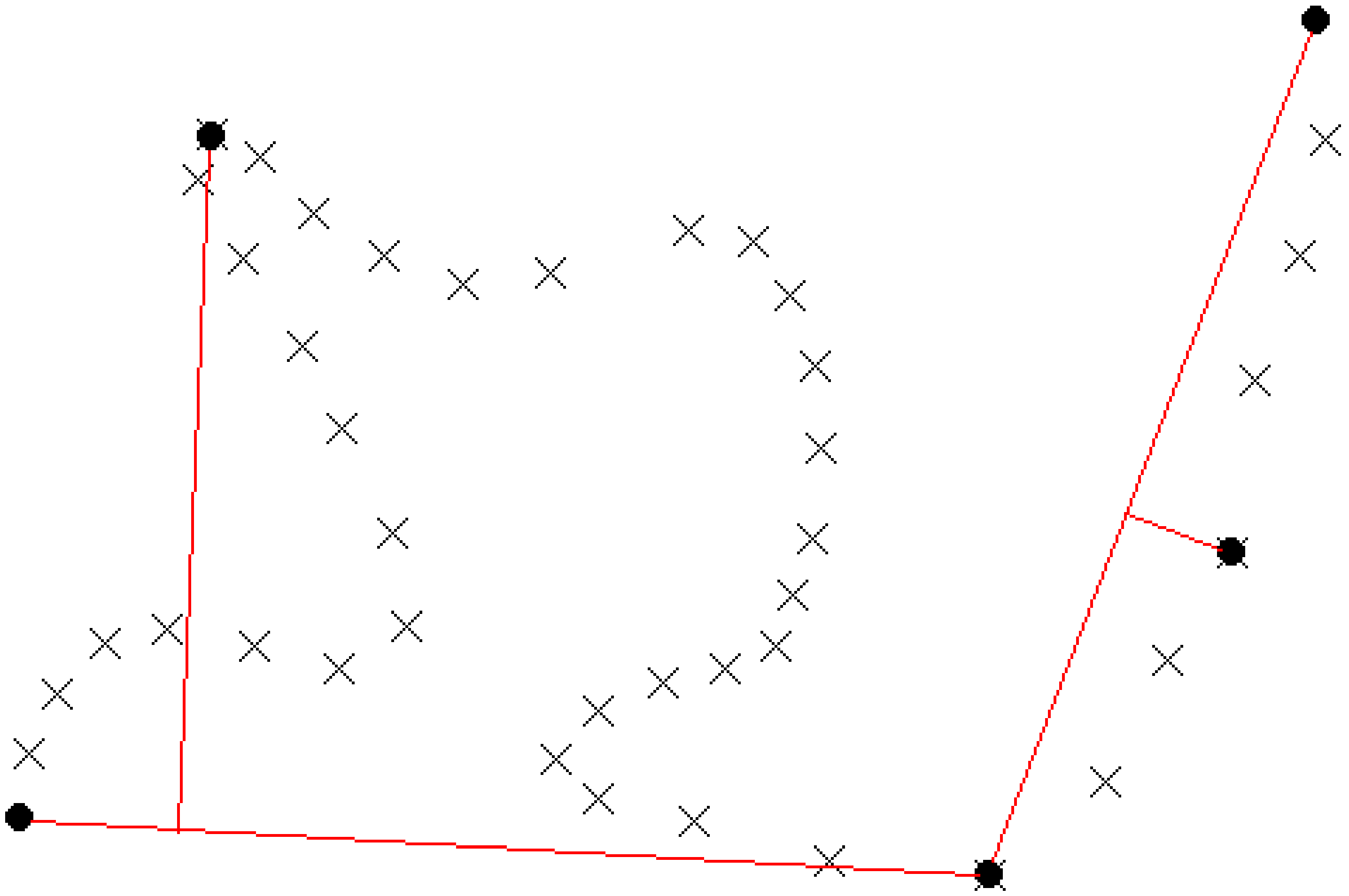
- reducing a point set by removal of vertices if they fall within a bandwidth tolerance.
- *progressive* subdivision of the polyline on either side of the vertex which lies furthest from the straight line between two end nodes of the sub-segment

Douglas-Peuker Line Generalisation



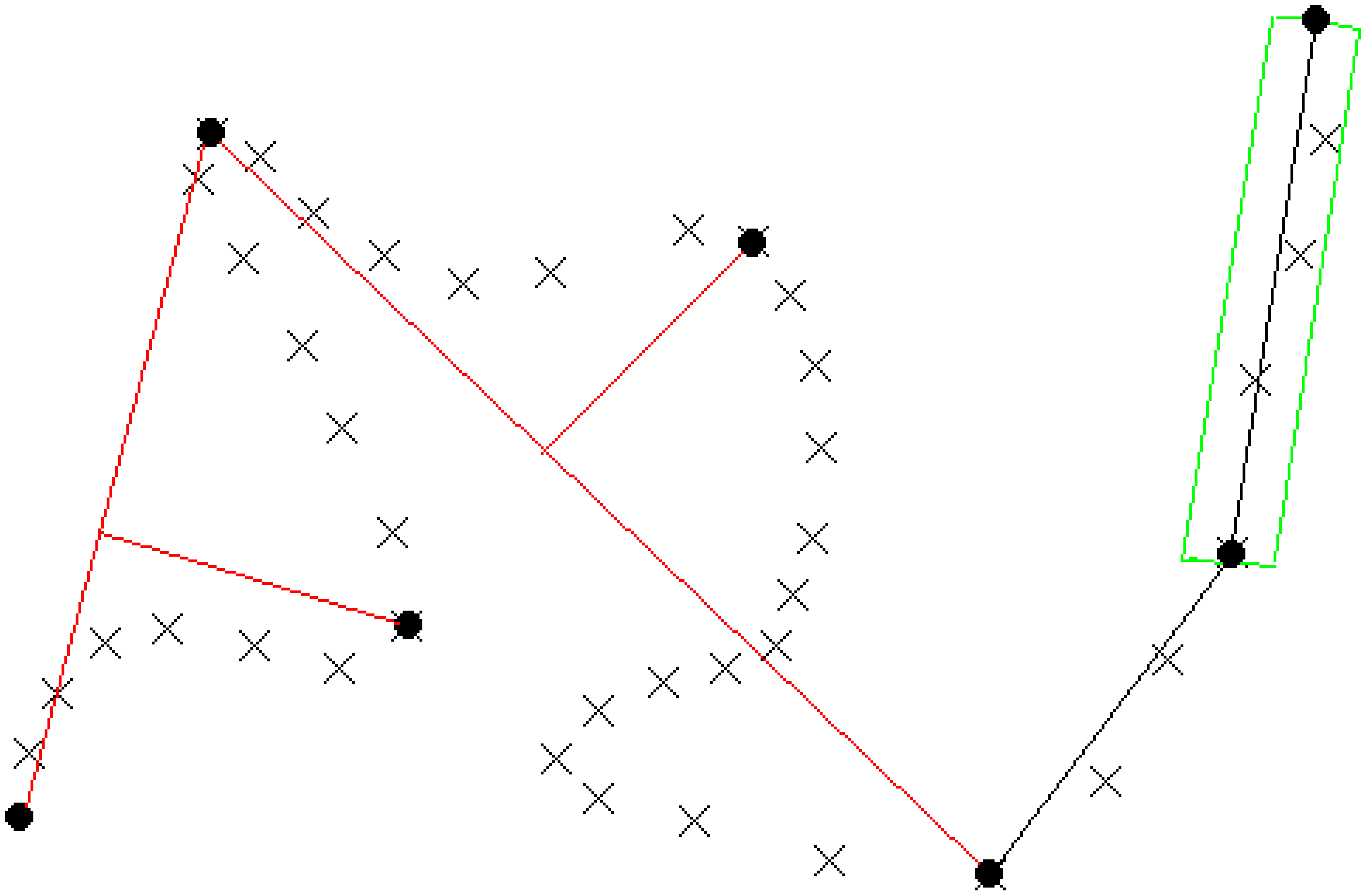
- Draw a straight line between start and end nodes.
- Locate vertex which lies at the greatest perpendicular distance from the straight line.
- Examine to see if this lies within the linear tolerance set.

Douglas-Peuker Line Generalisation



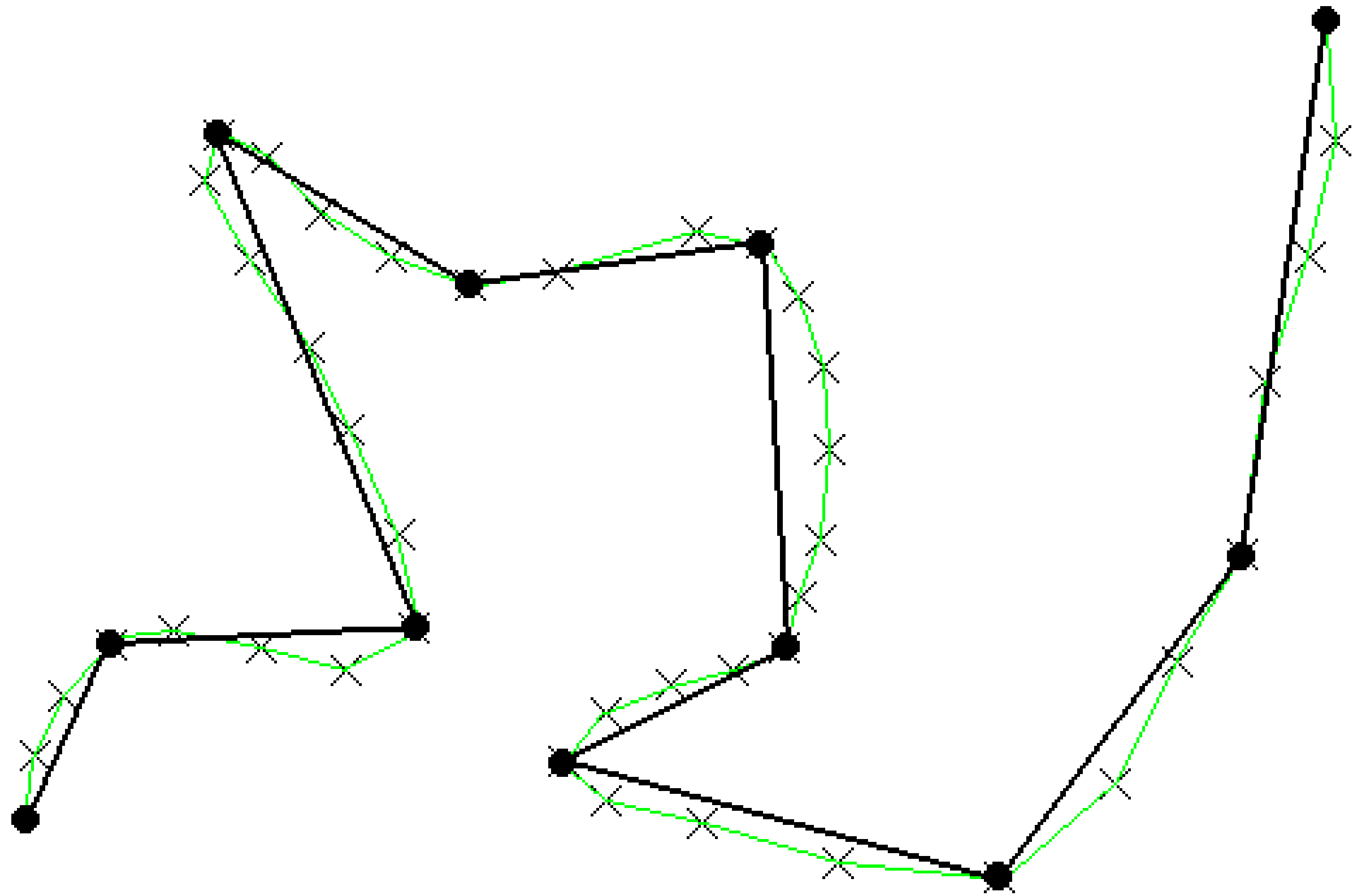
- … if not,
- repeat the process
- this time forming two new straight lines from the start node to the new node and from the new node to the end node.
- For each of these new lines now find the furthest point and examine to see if it lies within the set tolerance.

Douglas-Peuker Line Generalisation



- Repeat iteratively.
- If the points all fall within the tolerance set then proceed no further with this section of the arc.
- The remaining vertices are part of the desired generalisation

Douglas-Peuker Line Generalisation



… and finally.

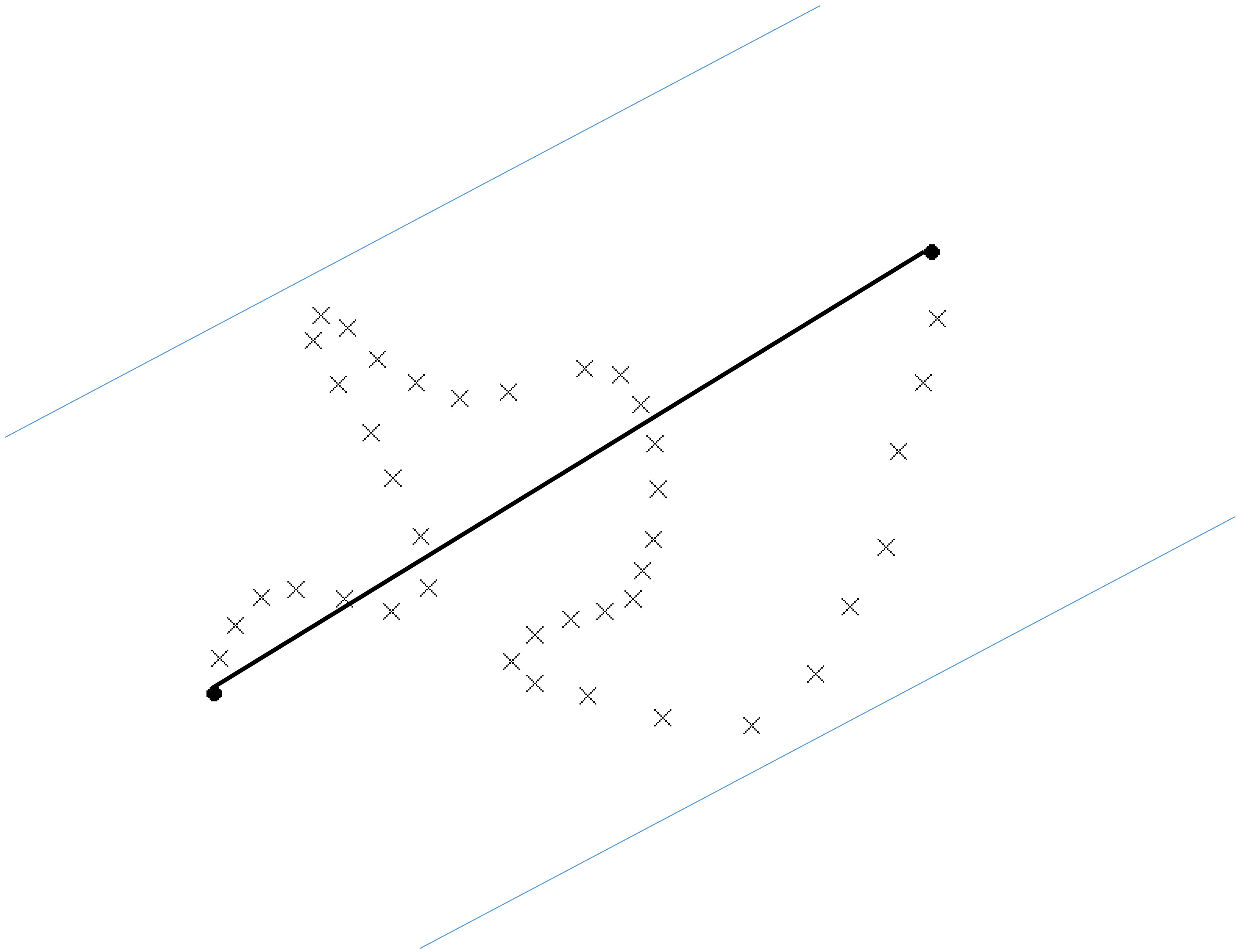
How do we do this in Python?

- Break it down:
    1. In English
    2. Pseudo-code
    3. Python code

In English the problem for generalisation

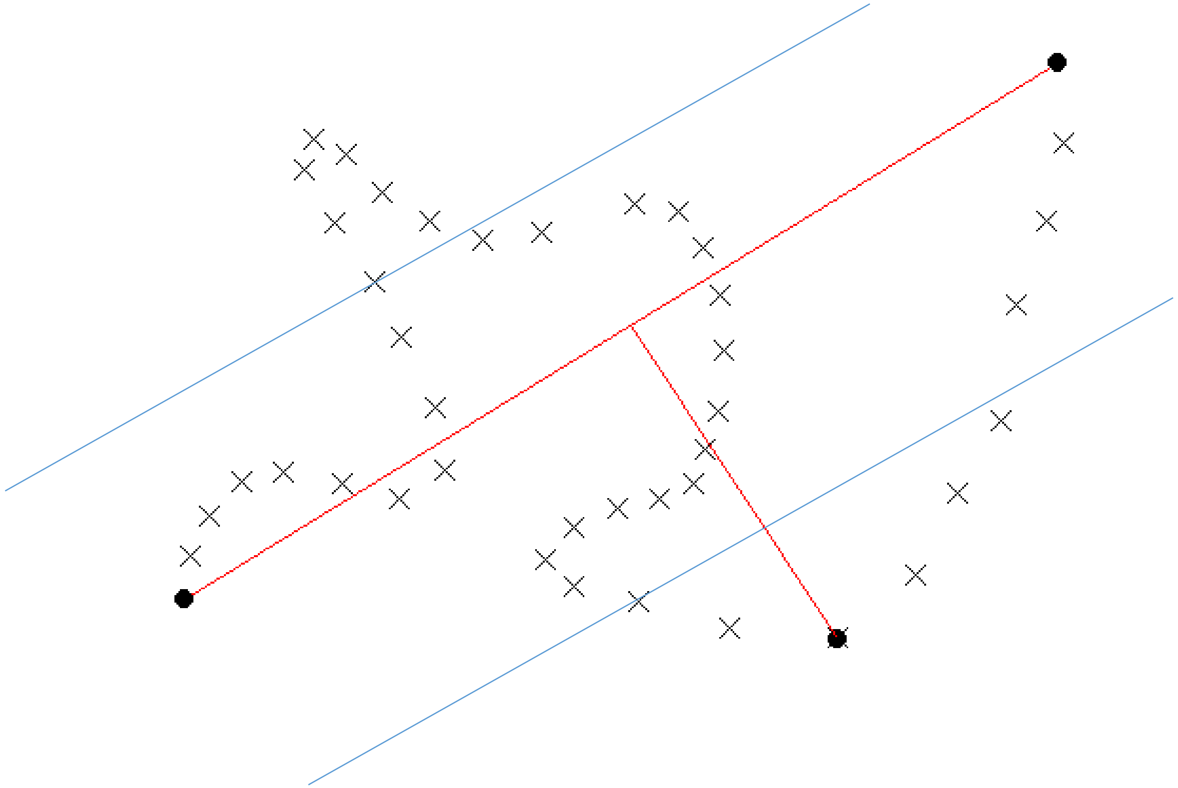
- For a polyline
  - Start with a polyline and return a polyline.
  - If only two points exist it is already simple enough so return the original line
- If more than 2 points exist
  - Construct a segment between endpoints
  - Go through each point remembering which point is further from the segment
  - If the furthest point is within-tolerance (t) return the segment as a polyline

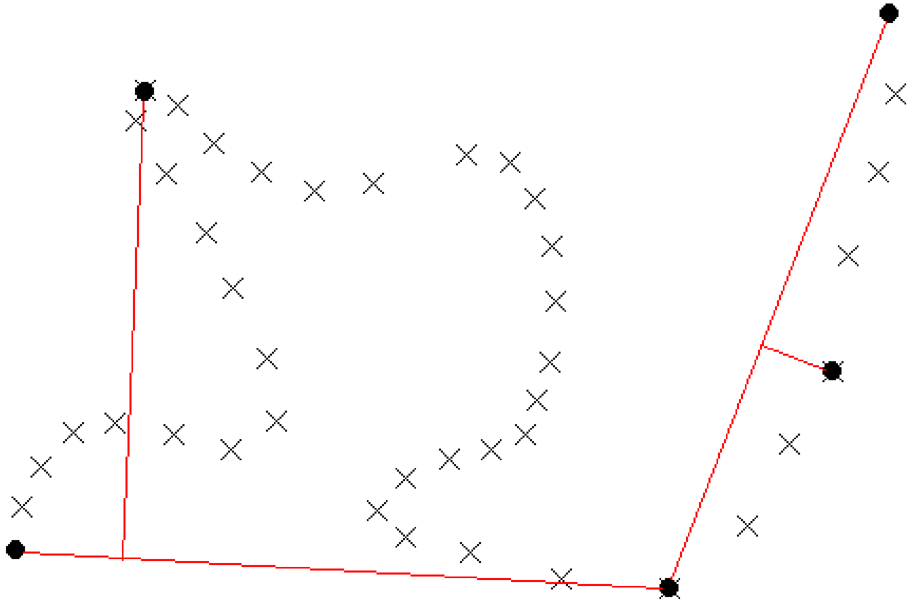
Tolerance not breached



If tolerance is breached

- If tolerance is breached:
  - Call the method again for two sub-polylines (c1 and c2)
  - Construct a polyline from the two sub-polylines and return this





Pseudo-code

- For a polyline
    - Start with polyline
    - If only two points in polyline
        - Return as polyline *#do nothing*
    - Else
        - Construct a segment between start and end point
        - For
            - Each point in the segment remember which is furthest from segment
            - If
                - Furthest point is within the tolerance
                - Return the start-end segment as a polyline *# will remove all points in between start and end*

Pseudo-code

- If
  - furthest point outside tolerance
  - Create new polyline 1
    - Start-point to furthest point in segment
  - Create new polyline 2
    - Furthest point in segment to end point
  - Call the method again for two sub-polylines *# recursion happening here*
    - Construct a polyline from the two sub-polylines
  - Return the new polyline

How to engineer this?
Convert pseudo-code to python code

Open
Polyline.py in spyder or what ever you use.

How do you engineer this?

- For a polyline:
  - If there are only two points in the polyline return it
    - Check how may points there are
    - Include a ‘size-of’ method in our polyline class
      - This size of method reports how many points there are in a polyline (chain) object.

```
    def generalise(self, t):
        if (self.size()<3):
            print ('No more points')
            return self
```

How do you engineer this?

- If more than 2
  - Construct a segment between endpoints
  - We can write a method of our polyline class, getStartEndSeg, that returns a LineSegment from the start to the end node of the polyline.

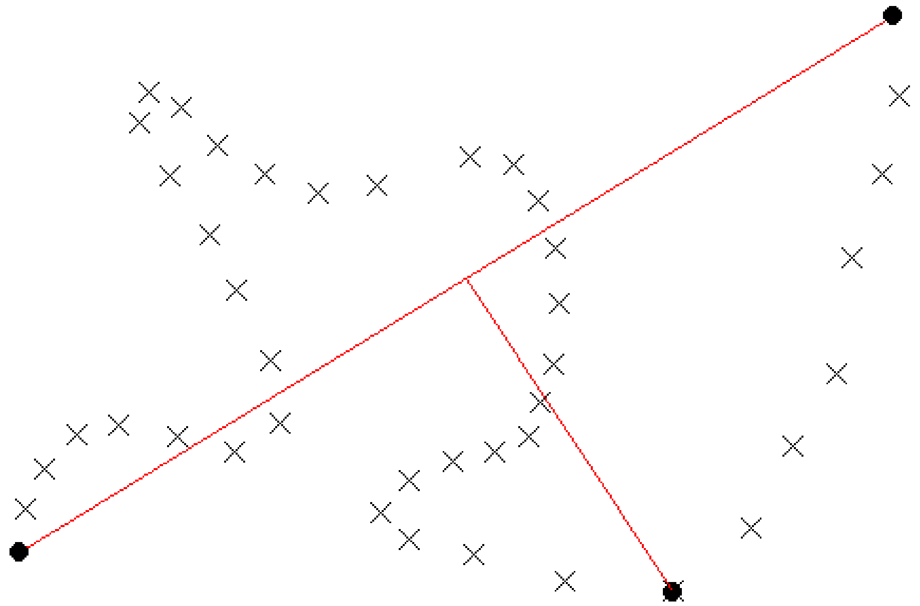
```
    def generalise(self, t):
        if (self.size()<3):
            print ('No more points')
            return self
        else:
            dp=self.furthestFromSeg()

            if (dp.getD()<t):
                print ('Within tolerance {}, max dist at {}'.format(t, dp))
                newSeg = self.getStartEndSeg()
                print ('returning {}'.format(newSeg))
                return newSeg.segAsPolyline()
```

What are these?

How do you engineer this?

- Go through each point remembering which point is further from the segment
- Now since there were more than two points in our chain, iterate through points 2 to (n-1), and see which one lies furthest from the segment.

- This requires three things:
  1. That we iterate through the points 2 to (n-1)
  2. That we have a method of our segment class that returns the point to segment distance
  3. That we remember what the largest distance is (i.e. we search for a maximum). Also it will be good to remember which point this is (its index).



How do you engineer this?

- If the furthest point is within tolerance, return the segment as a polyline
  - Test if our maximum distance from the start-end segment within the tolerance
  - We write a method of our segment class that returns a version of a segment as a two-point polyline object.

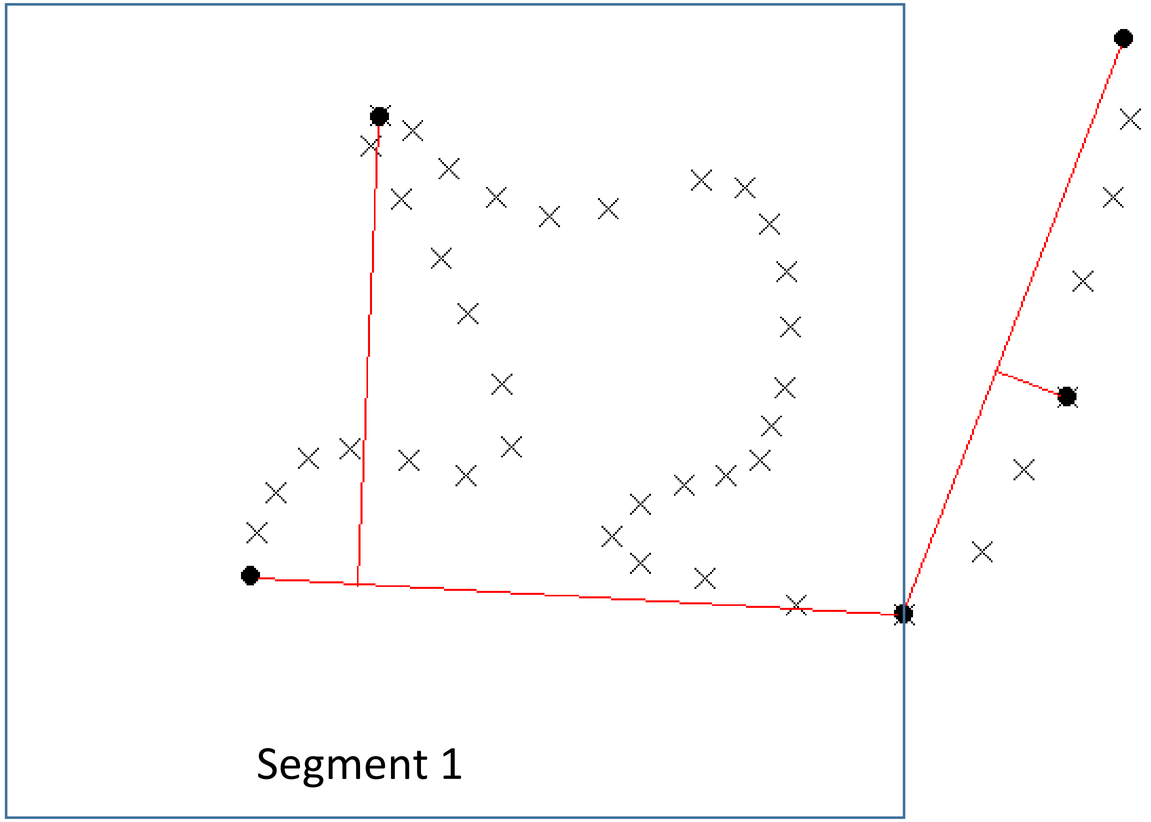
```
    def generalise(self, t):
        if (self.size()<3):
            print ('No more points')
            return self
        else:
            dp=self.furthestFromSeg()

            if (dp.getD()<t):
                print ('Within tolerance {}, max dist at {}'.format(t, dp))
                newSeg = self.getStartEndSeg()
                print ('returning {}'.format(newSeg))
                return newSeg.segAsPolyline()
```

How do you engineer this?

- If tolerance is breached:
  - Call the method again for two sub-polylines (c1 and c2)
  - This involves recursion
  - Everything we have done so far we can start again and treat the problem in two separate halves. Apply everything we have jus done to each separate half
  - Need to create two sub-polylines from the split – so remember the index number of the maximum separation point

```
        else:
            print ('Splitting at {}'.format(dp))
            v=self.split(dp.getI())

            c1=v[0]
            c2=v[1]

            c1=c1.generalise(t)
            c2=c2.generalise(t)

            return (self.combinePolyline(c1, c2))
```

How do you engineer this?



How do you engineer this?

- Construct a polyline from the two sub-polylines and return that
- Here we are rebuilding the separate results of the recursive procedure
- Write a function that accepts two existing polylines as arguments to create and return a new polyline
- In this case the middle point will be the same and we want to exclude it but more generically we probably want to check if the end of the first chain and start of the second chain are the same

```
    def generalise(self, t):
        if (self.size()<3):
            print ('No more points')
            return self
        else:
            dp=self.furthestFromSeg()

            if (dp.getD()<t):
                print ('Within tolerance {}, max dist at {}'.format(t, dp))
                newSeg = self.getStartEndSeg()
                print ('returning {}'.format(newSeg))
                return newSeg.segAsPolyline()

            else:
                print ('Splitting at {}'.format(dp))
                v=self.split(dp.getI())

                c1=v[0]
                c2=v[1]

                c1=c1.generalise(t)
                c2=c2.generalise(t)

                return (self.combinePolyline(c1, c2))
```

You have the module.
We defined a class called polyline and gave it some characteristics and some methods that it expects
The great thing about OOP is that you do not necessarily have to understand everything that is going on, just that it works and when you need to edit the code you can do.

This is quite a jump !

- Quite a jump from last week
- Lets go through the module again together…

Coding problem 5: Generalisation

```python
    def generalise(self, t):
        if (self.size()<3):
            print ('No more points')
            return self
        else:
            dp=self.furthestFromSeg()

            if (dp.getD()<t):
                print ('Within tolerance {}, max dist at {}'.format(t, dp))
                newSeg = self.getStartEndSeg()
                print ('returning {}'.format(newSeg))
                return newSeg.segAsPolyline()

            else:
                print ('Splitting at {}'.format(dp))
                v=self.split(dp.getI())

                c1=v[0]
                c2=v[1]

                c1=c1.generalise(t)
                c2=c2.generalise(t)

                return (self.combinePolyline(c1, c2))
```

- Classes and Objects
  - Need to include a ‘size-of’ method in the polyline class
  - Reports how many points are in a polyline (chain) object

```python
    def size(self):
        return len(self._allPoints)
```

Coding problem 5: Generalisation

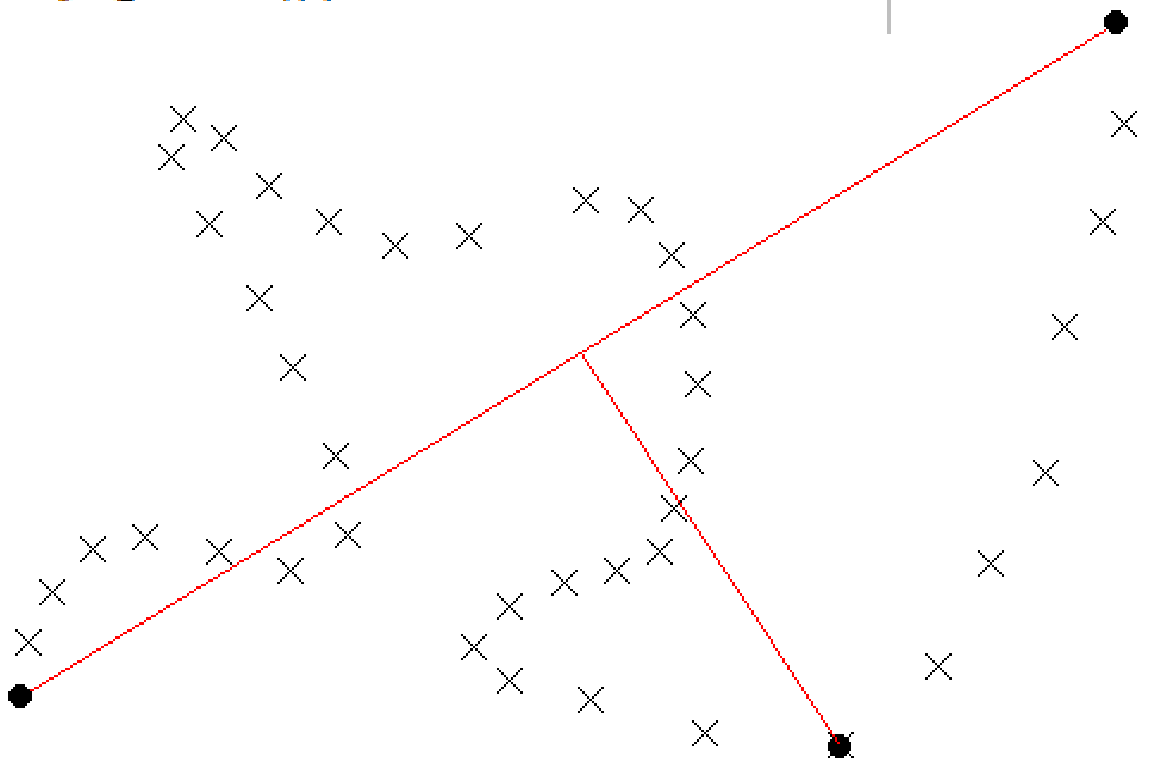
- Construct a segment between the end points
- Need to include a segment class
- Can include a method for polyline class:
  - getStartEndSeg
  - Which returns a LineSegment from the start to the end node of the polyline.

```python

    def getStartEndSeg(self):
        if (self.size()<2):
            return None
        else:
            return Segment(self.getStart(), self.getEnd())
```

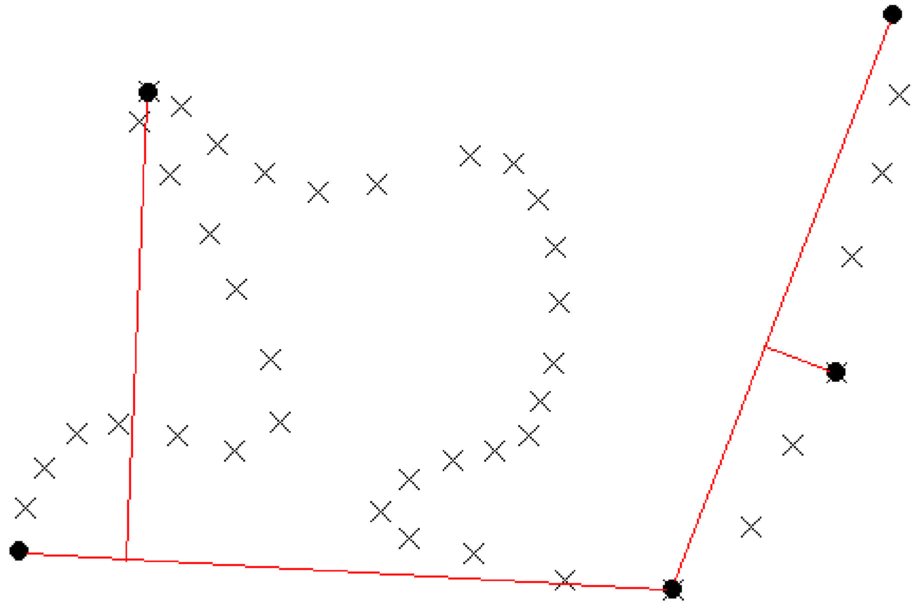
GetStartEndSeg

```python
    def getStartEndSeg(self):
        if (self.size()<2):
            return None
        else:
            return Segment(self.getStart(), self.getEnd())
```

```python
# Returns start Point
    def getStart(self):
        if (len(self._allPoints)>0):
            return self.getPoint(0)
        else:
            return None

# Returns end Point
    def getEnd(self):
        if (len(self._allPoints)>0):
            return self.getPoint(self.size()-1)
        else:
            return None
```



Recursion



```python

##implementation of Douglas-Peuker line generalisation
        # t is bandwidth for the algorithm specifying the level of
    def generalise(self, t):
        #will only work if there are more than 2 points otherwise
        if (self.size()<3):
            print ('No more points')
            return self
        else:
#get the furtheest point - goes through all points to find the fur
            dp=self.furthestFromSeg()
# if the further point lies with the bandwidth we can reduce this chain to the end segment
            if (dp.getD()<t):
                print ('Within tolerance {}, max dist at {}'.format(t, dp))
                newSeg = self.getStartEndSeg()
                print ('returning {}'.format(newSeg))
# so can return this segment as a chain
                return newSeg.segAsPolyline()
#otherwise
            else:
#split the chain at the furthest point (DP holds the index point)
                print ('Splitting at {}'.format(dp))
                v=self.split(dp.getI())
# extract the two new chains independently from the vector returned above
                c1=v[0]
                c2=v[1]
#no the recursive bit - you can generalise these two
                ##each part creates a generalised version of the sub-chain
                c1=c1.generalise(t)
                c2=c2.generalise(t)
##combine the two generalised sub-chains
                return (self.combinePolyline(c1, c2))

```

Check there’s more than 2 points in this chain

```
##implementation of Douglas-Peuker line generalisation
        # t is bandwidth for the algorithm specifying the level of generalisation
    def generalise(self, t):
        #will only work if there are more than 2 points otherwise it returns the original
        if (self.size()<3):
            print ('No more points')
            return self
        else:
#get the furtheest point - goes through all points to find the furt
            dp=self.furthestFromSeg()
# if the further point lies with the bandwidth we can reduce this chain to the end segment
            if (dp.getD()<t):
                print ('Within tolerance {}, max dist at {}'.format(t, dp))
                newSeg = self.getStartEndSeg()
                print ('returning {}'.format(newSeg))
# so can return this segment as a chain
                return newSeg.segAsPolyline()
#otherwise
            else:
#split the chain at the furthest point (DP holds the index point)
                print ('Splitting at {}'.format(dp))
                v=self.split(dp.getI())
# extract the two new chains independently from the vector returned above
                c1=v[0]
                c2=v[1]
#no the recursive bit - you can generalise these two
                ##each part creates a generalised version of the sub-chain
                c1=c1.generalise(t)
                c2=c2.generalise(t)
##combine the two generalised sub-chains
                return (self.combinePolyline(c1, c2))
```

Go through all the other points to see which is furthest from the start-end segment

```python
##implementation of Douglas-Peuker line generalisation
        # t is bandwidth for the algorithm specifying the level of generalisation
    def generalise(self, t):
        #will only work if there are more than 2 points otherwise it returns the original
        if (self.size()<3):
            print ('No more points')
            return self
        else:
#get the furtheest point - goes through all points to find the furthest from the start-end s
            dp=self.furthestFromSeg()
# if the further point lies with the band
            if (dp.getD()<t):
                print ('Within tolerance {}, max dist at {}'.format(t, dp))
                newSeg = self.getStartEndSeg()
                print ('returning {}'.format(newSeg))
# so can return this segment as a chain
                return newSeg.segAsPolyline()
#otherwise
            else:
#split the chain at the furthest point (DP holds the index point)
                print ('Splitting at {}'.format(dp))
                v=self.split(dp.getI())
# extract the two new chains independently from the vector returned above
                c1=v[0]
                c2=v[1]
#no the recursive bit - you can generalise these two
                ##each part creates a generalised version of the sub-chain
                c1=c1.generalise(t)
                c2=c2.generalise(t)
##combine the two generalised sub-chains
                return (self.combinePolyline(c1, c2))
```

If the furthest point is within-distance treat the start-end segment as a chain and return that

```python

##implementation of Douglas-Peuker line generalisation
        # t is bandwidth for the algorithm specifying the level of generalisation
    def generalise(self, t):
        #will only work if there are more than 2 points otherwise it returns the original
        if (self.size()<3):
            print ('No more points')
            return self
        else:
#get the furtheest point - goes through all points to find the furthest from the start-end s
            dp=self.furthestFromSeg()
# if the further point lies with the bandwidth we can reduce this chain to the end segment
            if (dp.getD()<t):
                print ('Within tolerance {}, max dist at {}'.format(t, dp))
                newSeg = self.getStartEndSeg()
                print ('returning {}'.format(newSeg))
# so can return this segment as a chain
                return newSeg.segAsPolyline()
#otherwise
            else:
#split the chain at the furthest point (DP holds the index poi
                print ('Splitting at {}'.format(dp))
                v=self.split(dp.getI())
# extract the two new chains independently from the vector returned above
                c1=v[0]
                c2=v[1]
#no the recursive bit - you can generalise these two
                ##each part creates a generalised version of the sub-chain
                c1=c1.generalise(t)
                c2=c2.generalise(t)
##combine the two generalised sub-chains
                return (self.combinePolyline(c1, c2))

```

Otherwise, make two new chains split at the point which is furthest, and repeat process on these….(recursion)

```python

##implementation of Douglas-Peuker line generalisation
        # t is bandwidth for the algorithm specifying the level of generalisation
    def generalise(self, t):
        #will only work if there are more than 2 points otherwise it returns the original
        if (self.size()<3):
            print ('No more points')
            return self
        else:
#get the furtheest point - goes through all points to find the furthest from the start-end s
            dp=self.furthestFromSeg()
# if the further point lies with the bandwidth we can reduce this chain to the end segment
            if (dp.getD()<t):
                print ('Within tolerance {}, max dist at {}'.format(t, dp))
                newSeg = self.getStartEndSeg()
                print ('returning {}'.format(newSeg))
# so can return this segment as a chain
                return newSeg.segAsPolyline()
#otherwise
            else:
#split the chain at the furthest point (DP holds the index point)
                print ('Splitting at {}'.format(dp))
                v=self.split(dp.getI())
# extract the two new chains independently from the vector returned above
                c1=v[0]
                c2=v[1]
#no the recursive bit - you can generalise these two
                ##each part creates a generalised version o
                c1=c1.generalise(t)
                c2=c2.generalise(t)
##combine the two generalised sub-chains
                return (self.combinePolyline(c1, c2))

```

Lastly, return a new chain made up from the results of the simplified sub-chains

Algorithm Design

- The method itself is relatively simple given the complexity of the task because we've given a lot of the work to other methods. ***This is the essence of good design***
- **Specifically we need……**
- A Segment class with a `pointDist(Point)method` to return the distance of a point from a line segment
- **To also have in our `Polyline` class…**
- To access the number of points in the Polyline `self.size()`
- A method `getStartEndSeg()` to return a line segment that spans the start and end points of our Polyline
- A method to create a sub-chains of our Polyline from any two or more existing points in the chain (splits it)
- A function that creates a Polyline from two existing Polylines by combining them together.

Any Questions?

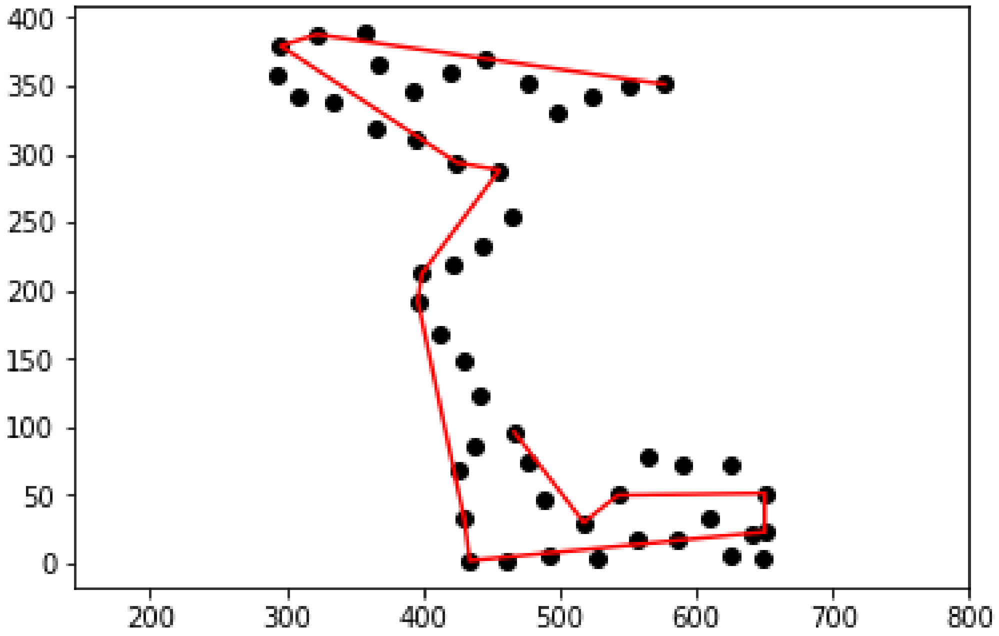
Learning Diary Task

- You have been provided with a driver, some modules and some test datasets
- We want you to perform a line generalisation
- However, the code isn't complete.
- Using the descriptions above
  - we want you to identify the code required
  - and to add them to the correct position in the modules.
- **Try to make the algorithm work, but if you cannot don't worry. Use Do as much as you can and provide English and Pseudo-code to show your working for the bits that you cannot work out.**

Task

- Hint: Running the driver immediately will give you a starting point
- We want you to, develop the polyline class
- Some of the functionality has been discussed in the lecture:
  - Identify start and end points of a segment
  - Generalisation method
- There are a couple of additional functions that you will need to add that we have not discussed in detail but mention in the English description:
  - An appropriate way of finding the furthest point on a *polyline* from a segment connecting the ends
  - Suitable method to split a Polyline at a specific index node, returning two separate Polylines.

Output should look something like this;



It will look different as we have not given you different wiggle files to test.

Task

```
from PointPlotter import PointPlotter
from ChainHandler import ChainLoader

xlo=0.0
xhi=1000.0
ylo=0.0
yhi=1000.0


pp=PointPlotter()
pp.set_axis(xlo, xhi, ylo, yhi)

chain=ChainLoader("Wiggle1.txt")
pp.plotPoint(chain[0]._allPoints, 'black')

pp.plotPolylines(chain[0].generalise(40.0), 'red')
pp.show()
```

- Lecture2D.py gives framework for loading and displaying test polylines

What to upload to the learning diary

- If you complete the task,
  - upload a brief description (<500 words) of the code you added and where you put it and include an image of the output.
  - Zip all of the modules, the driver and the wiggle.txt file that you used for testing and upload them to the diary – i should be able to run the driver and get the same output as you provide.
- If you do not complete the task by Saturday at noon:
  - Upload a brief description of the code that you did produce and where you put it. Include any thoughts on what else you need to do and where it would go. Pseudo-code could be helpful to support the english.
  - Zip all of the modules, the driver and the wiggle.txt file you used for testing with comments