WEEK 7 : CLASSES, OBJECTS AND USER-TYPES :

Lecture 1: Abstract datatypes, classes and objects:

# Lecture 2: Classes and objects in Python

\_\_init\_\_ = constructor, called when object is created.

\_\_str\_\_ =

* return string representation of object.
* Str(o) = o.\_\_str\_\_()
* Implicitly invoked by print()
* Def \_\_str\_\_(self):

Return (‘self.x ,self.y’)

\_\_add\_\_() :

* Invoked implicitly by +
* P1 + p2 = p1.\_\_add\_\_(p2)

\_\_mult\_\_():

Called implicitly by \*

\_\_lt\_\_() : less than : called implicitly by <

\_\_gt\_\_() : greater than : called implicitly by >

\_\_le\_\_() : less than equal to : <=

# Lecture 3: User defined lists :

# List is sequence of nodes.

class Node:  
 def \_\_init\_\_(self,v):  
 self.value = v  
 self.next = None  
 return  
  
 def \_\_str\_\_(self):  
 selflist = []  
 if self.value == None:  
 return str(selflist)  
  
 temp = self  
 selflist.append(self.value)  
  
 while (temp.next != None):  
 temp = temp.next  
 selflist.append(temp.value)  
  
 return (str(selflist))  
  
 def isempty(self):  
 if self.value == None:  
 return True  
 else:  
 return False  
  
 def append(self,v):  
 if self.isempty():  
 self.value = v  
 elif self.next == None:  
 newnode = Node(v)  
 self.next = newnode  
 else:  
 self.next.append(v)  
 return  
  
 def insert(self,v):  
 if self.isempty():  
 self.value = v  
 return  
  
 newnode = Node(v)  
  
 self.value, newnode.value = newnode.value, self.value  
 self.next, newnode.next = newnode.next, self.next  
  
 def delete(self,v):  
 if self.isempty():  
 return  
  
 if self.value == v:  
 self.value = None  
 if self.next != None:  
 self.value = self.next.value  
 self.next = self.next.next  
 return  
 else:  
 if self.next != None:  
 self.next.delete(v)  
 if self.next.value == None:  
 self.next = None  
 return  
  
n = Node(10)  
print(n)  
n.append(20)  
print(n)  
n.delete(20)  
print(n)  
n.insert(50)  
print(n)  
print(n.isempty())

# Lecture 4: Search trees

# Binary search tree :

# Left child : value less than parent node

# Right child : value greater than parent node

# 

Repeat this

def foo(self):

if self.isempty():

return(0)

elif self.isleaf():

return(1)

else:

return(self.left.foo() + self.right.foo()))

ans : The number of leaves in mytree.

def foo(self):

if self.isempty():

return(0)

elif self.isleaf():

return(1)

else:

return(1 + max(self.left.foo(),self.right.foo()))

ans : length of longest path from root to leaf in mytree.