Introduction to Trees

Chapter 6

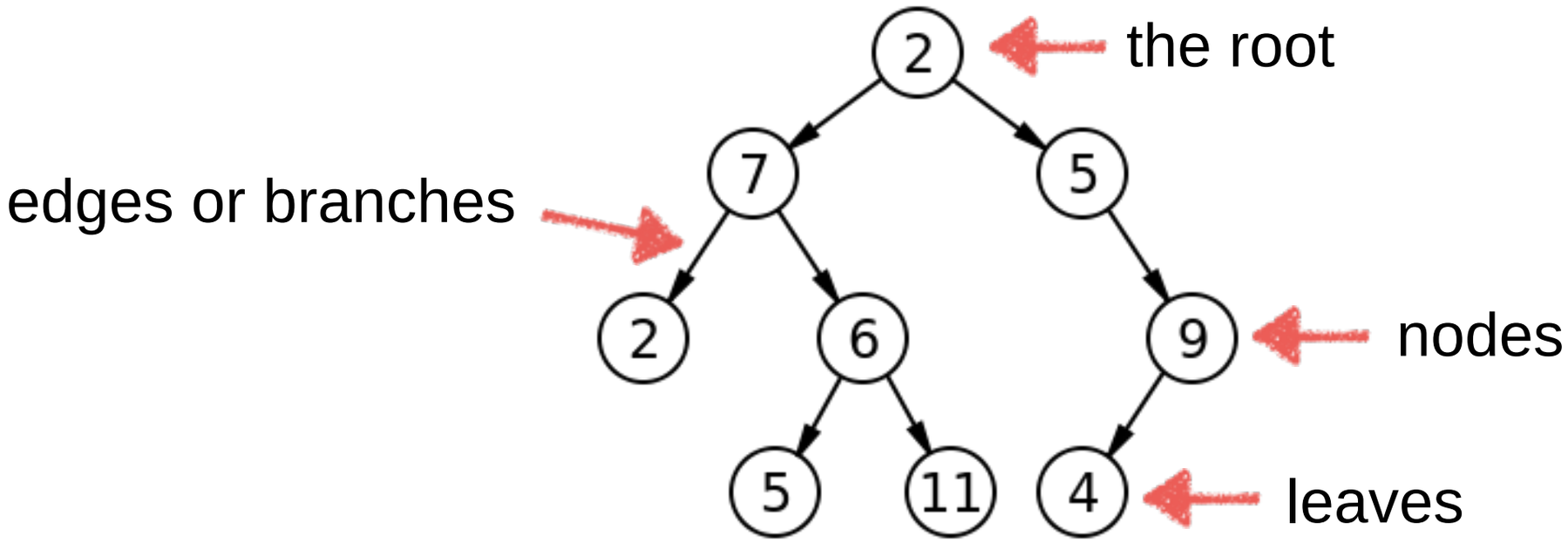
Why Trees?

- Trees are amazingly useful in Computer Science.
  - They provide a structure for the efficient storage, manipulation and retrieval of information.
  - They map to many of the relationships between objects or classification systems in the real world.
  - They also give an introduction to Graph Theory (much more in 220 and 320).

By Ptelea (Own work) [CC-BY-SA-3.0 (http://creativecommons.org/licenses/by-sa/3.0)], via Wikimedia Commons

Different Types of Trees

- But the sort we will be working with are really *rooted trees* (they are directed graphs) and look like this:



http://en.wikipedia.org/wiki/File:Binary_tree.svg

What is Special?

- Our trees are upside down. The root is at the top and the leaves are at the bottom.

  - The only reason for this is that we normally write and read from the top of the page to the bottom.

    - It is easier to add things to the bottom of our diagrams.

    - We start at the root and move down the tree.

- Usually our trees will be binary trees (a maximum of two branches from a node).

Example - 20 questions

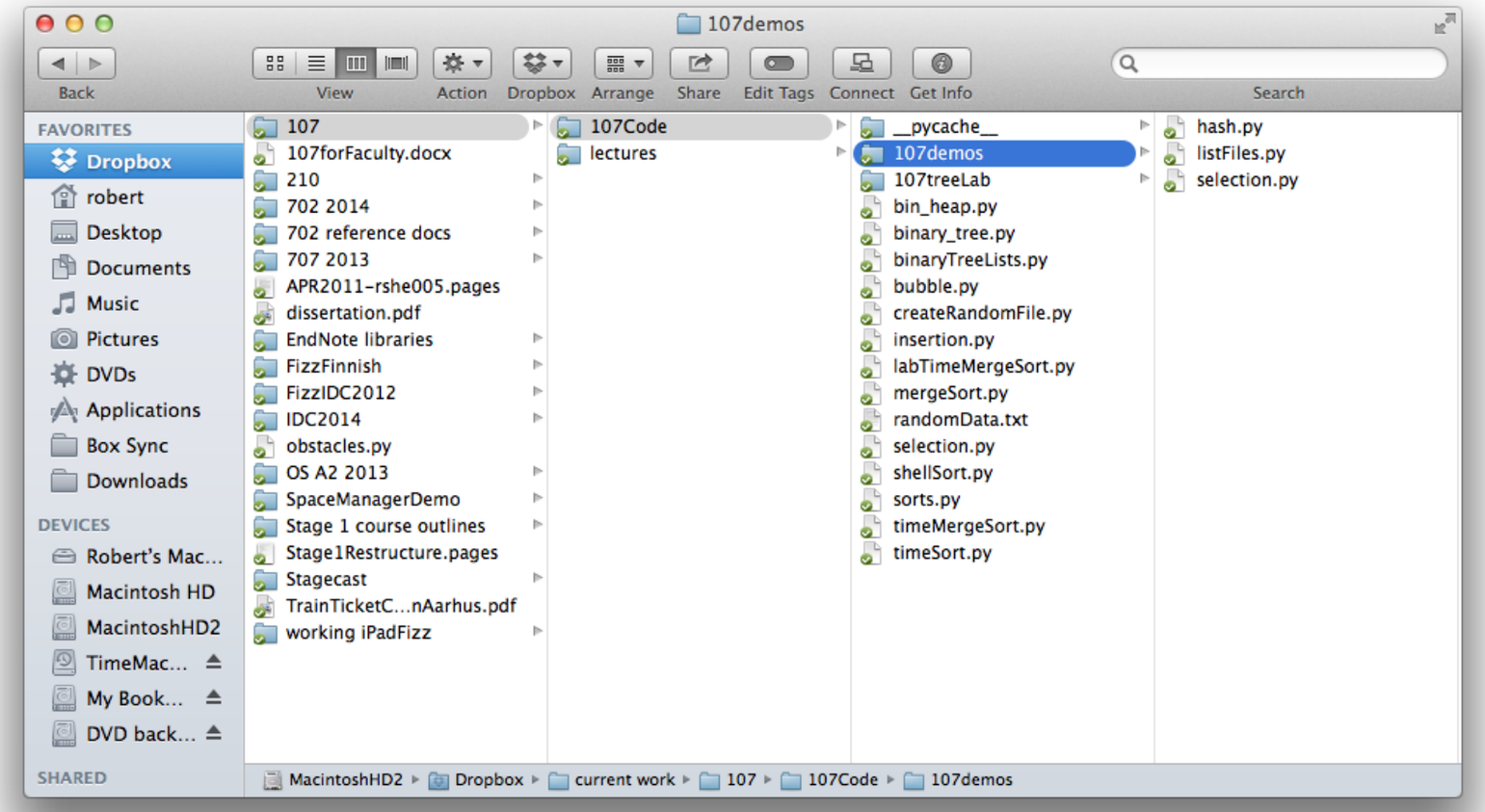
- One person chooses a subject and answers 20 questions with “yes” or “no” answers.

- This is an example of following one branch of a binary tree as each question leads to only two possible answers.

- In theory (with well designed questions) this can lead to differentiating 2^{20} (over a million) different subjects.

Example - file systems

- Most file systems are trees (folders or directories and files)



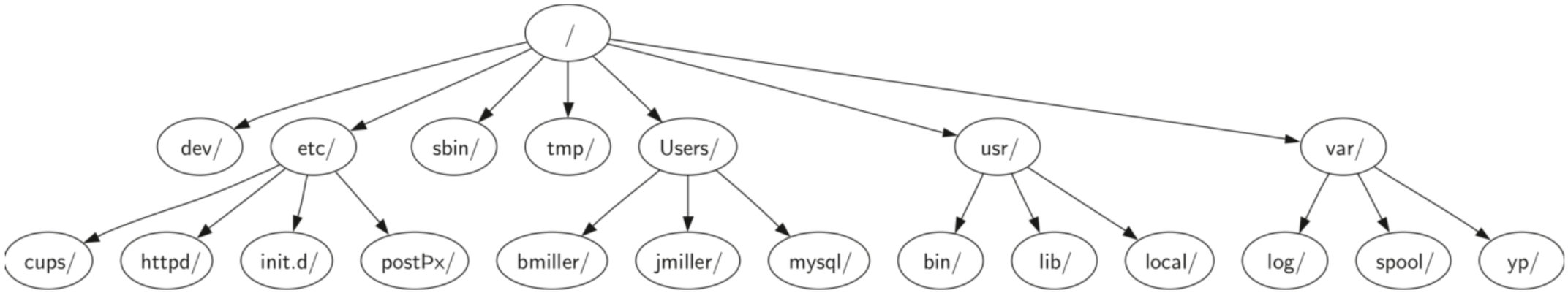
- Note the line at the bottom - this shows the *path* to the current directory.
  - There is a unique path from the root to the leaf.
  - This tree is not a binary tree (many files and directories can be descendants of one directory)

Subtrees

- Because all nodes in a tree (except the root node) are descendants of exactly one other node we can recursively think of each node being the root node of a smaller tree.

- These smaller trees are called subtrees.

- e.g. the Users/ node is the root of a subtree of users' directories in the diagram below (from the textbook)



XML and HTML

```
<html xmlns="http://www.w3.org/1999/xhtml"
    xml:lang="en" lang="en">
<head>
   <meta http-equiv="Content-Type"
        content="text/html; charset=utf-8" />
   <title>simple</title>
</head>
<body>
<h1>A simple web page</h1>
<ul>
   <li>List item one</li>
   <li>List item two</li>
</ul>
<h2><a href="http://www.cs.luther.edu">Luther CS </a><h2
</body>
</html>
```

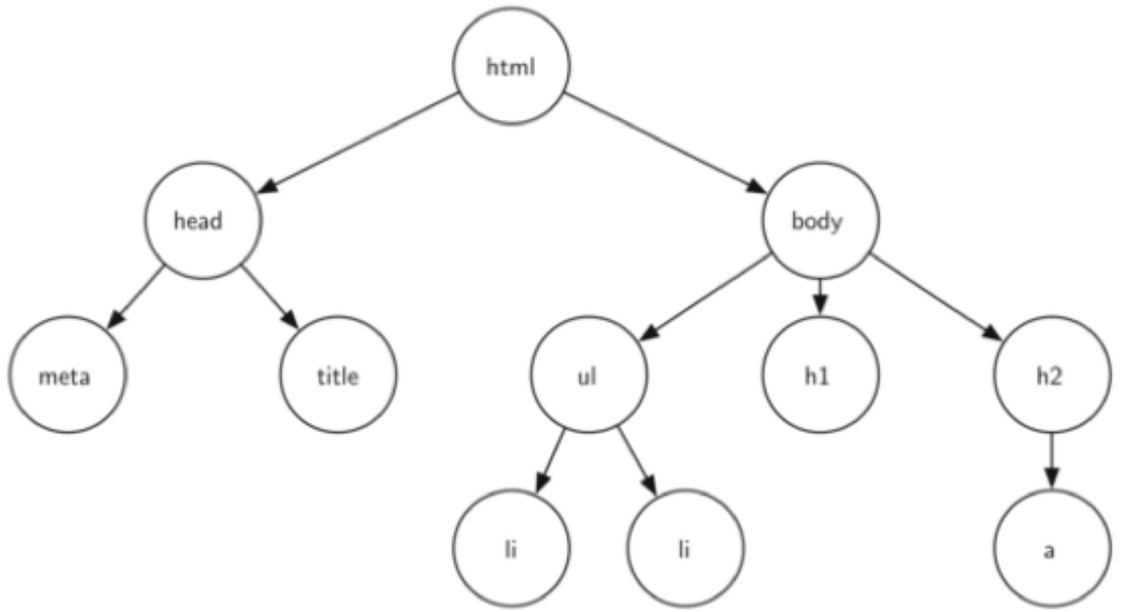


Figure 6.3: A Tree Corresponding to the Markup Elements of a Web Page

More Terminology

- Parent - a node is the parent of all nodes it connects to with outgoing edges.

- Children - all nodes with incoming edges from the same parent node are children of that node.

- Siblings - all nodes with the same parent are siblings.

- Level - the number of edges from the root node to this node. The root node is therefore level 0.

- Height - the height of a tree is the maximum level of all the nodes in the tree.

Tree definition 1

- A tree
  - has a root node
  - every node (except the root node) is connected by one edge from its unique parent node
  - a unique path goes from the root to each node
  - (and remember that a binary tree is one where the maximum number of children from a node is 2)

Tree definition 2

- A tree
  - can be empty
  - or consists of a root node and zero or more subtrees (subtrees are trees)
  - the root of a subtree is connected to the root of the tree by an edge
  - (a binary tree has no more than 2 subtrees from any root)

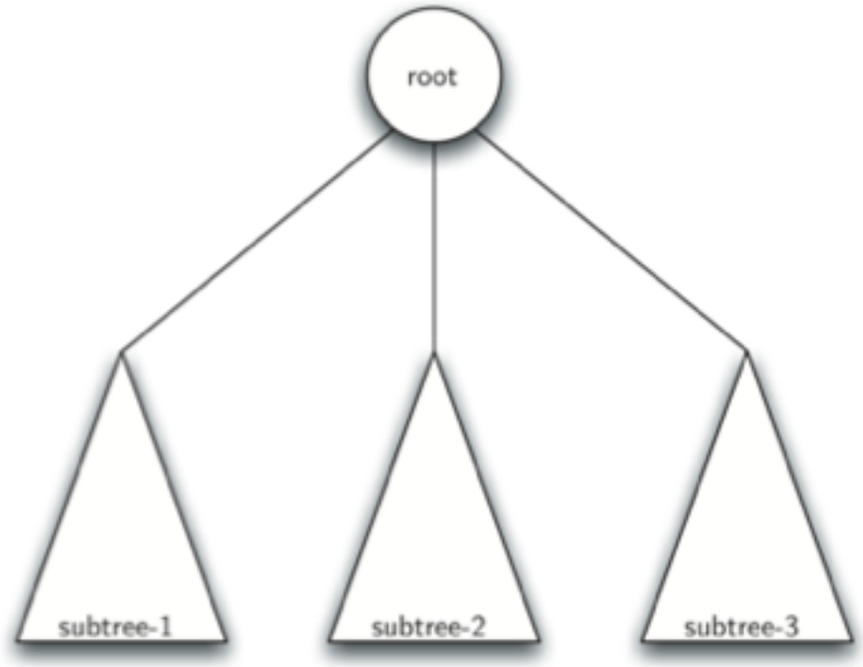


Figure 6.5: A recursive Definition of a tree

Binary Trees

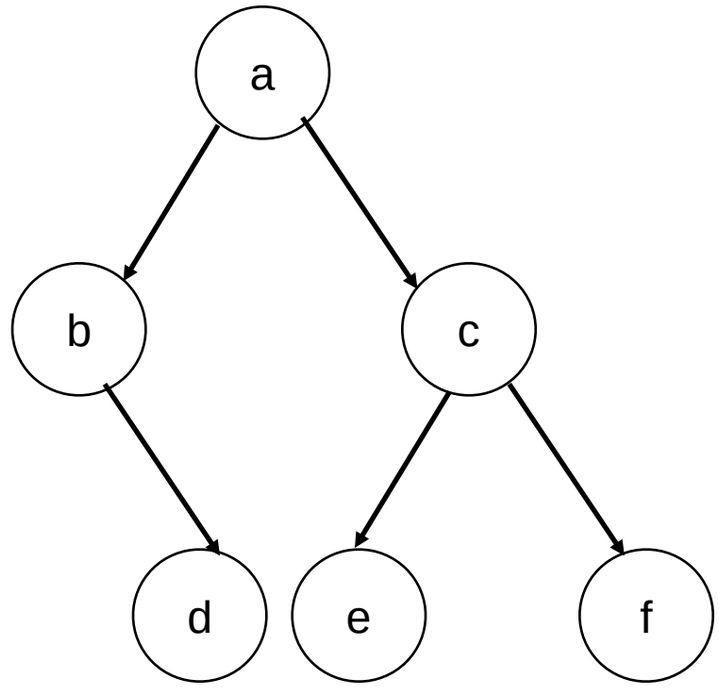
- From now on our trees will be binary trees (except in the lab questions)
- They are simple to work with as each parent node has left and/or right values.
- We can represent this something like:

```
class BinaryTree():

    def __init__(self, root_data):
        self.data = root_data
        self.left = None
        self.right = None
```

But we can simply use lists

- Before we develop our binary tree class with links to left and right subtrees we could simply use Python lists.
- e.g. this binary tree can be represented by the list




```
['a',
    ['b',
        [],
        ['d',
            [],
            []
        ]
    ],
    ['c',
        ['e',
            [],
            []
        ],
        ['f',
            [],
            []
        ]
    ]
]
```

And of course this list is actually just:

```
['a', ['b', [], ['d', [], []]], ['c', ['e', [], []], ['f', [], []]]]
```

[data, left, right]

- Each node in the tree is a list of three elements
- the data or value or payload or key
- the left subtree (which is another list)
- the right subtree (which is also a list)

Binary Tree ADT

- Some tree operations
  - BinaryTree() - create a new BinaryTree
  - set_value(new_value) - sets the value of the node to new_value
  - get_value() - gets the value of the node
  - insert_left(value) - creates a new node with value to the left of the current node, the existing left node becomes the left node of the new one
  - insert_right(value) - as above but to the right
  - get_left_subtree() - gets the subtree to the left of this node
  - get_right_subtree() - gets the subtree to the right of this node
- N.B. Using the recursive definition each node is the root of a subtree.

My Version

See binaryTreeLists.py and compare with the textbooks 6.4.1

```
class BinaryTree:

   DATA = 0     # just to make things more readable
   LEFT = 1     # can be referenced as either
   RIGHT = 2    # e.g. BinaryTree.DATA or self.DATA

   def __init__(self, root_value, left=None, right=None):
      self.node = [root_value, left, right]
```

The default values for “left” and “right” mean that the constructor can be called with only the value for the root node. In this case the left and right subtrees are empty.

Inserting

```
# Theirs (I don’t bother showing the insert_right)
def insert_left(root, new_branch):
    t = root.pop(1)
    if len(t) > 0:
        root.insert(1, [new_branch, t, []])
    else:
        root.insert(1, [new_branch, [], []])
    return root

# Mine
    def insert_left(self, value):
        self.node[self.LEFT] = BinaryTree(value, self.node[self.LEFT], None)

    def insert_right(self, value):
        self.node[self.RIGHT] = BinaryTree(value, None, self.node[self.RIGHT])
```

The other functions are straightforward

```
def set_value(self, new_value):
    """Sets the value of the node."""
    self.node[self.DATA] = new_value

def get_value(self):
    """Gets the value of the node."""
    return self.node[self.DATA]

def get_left_subtree(self):
    """Gets the left subtree of the node."""
    return self.node[self.LEFT]

def get_right_subtree(self):
    """Gets the right subtree of the node."""
    return self.node[self.RIGHT]
```

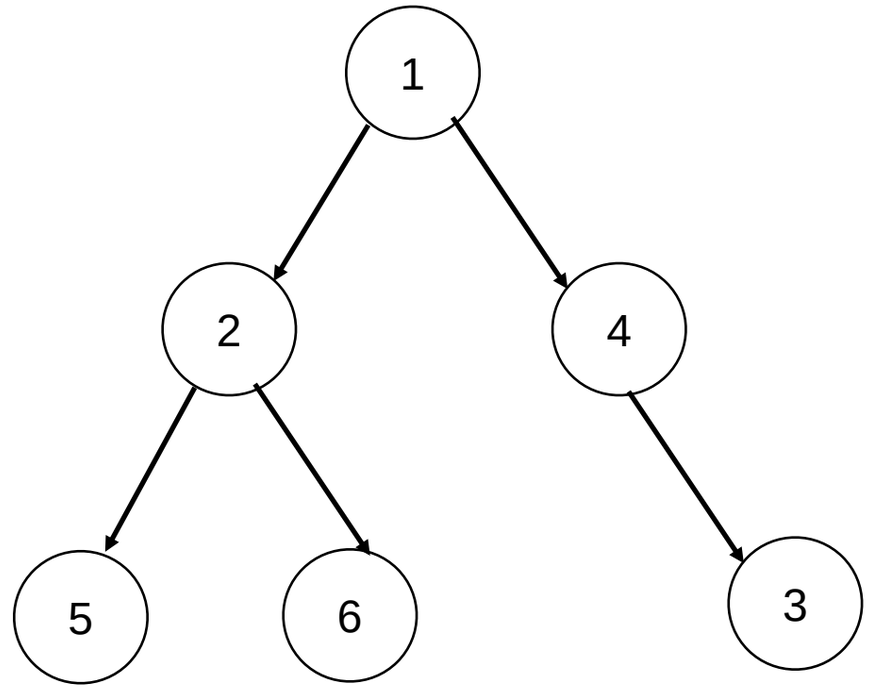
Printing using str()

Printing is easier in the textbook version because it just prints a list.
But my way allows me to use recursion and the __str__ special function.
Remember that __str__(self) is called when the str() function is used e.g. by the print() function.

```
def __str__(self):
   return '['+str(self.node[self.DATA])+', '+\
          str(self.node[self.LEFT])+', '+\
          str(self.node[self.RIGHT])+']'
```

Output

```
r = BinaryTree(1)
r.insert_left(2)
r.insert_right(3)
r.insert_right(4)
r.get_left_subtree().insert_left(5)
r.get_left_subtree().insert_right(6)
print(r)
print(r.get_left_subtree())
print(r.get_right_subtree())
print(r.get_left_subtree().get_left_subtree())
```



Produces

```
[1, [2, [5, None, None], [6, None, None]], [4, None, [3, None, None]]]
[2, [5, None, None], [6, None, None]]
[4, None, [3, None, None]]
[5, None, None]
```

K.I.S.S.

- Normally if you can use a built-in or standard Python data type to represent your data you should.
- Or as we just did create a new class with Python standard types as “instance variables” of the class.
- Sometimes you may subclass an existing class or data type to provide additional behaviour but this is beyond the scope of this course.
- With the BinaryTree class we could use a completely different implementation which doesn’t rely on Python lists.

Nodes and References

```
# See binaryTreeRef.py and once again compare with the
# textbook version of BinaryTree.

class MyBinaryTree:

    def __init__(self, root_key, left=None, right=None):
        self.key = root_key
        self.left = left
        self.right = right
```

Once again the default values mean that we can call this in multiple ways. e.g.

```
MyBinaryTree(3)
MyBinaryTree(3, existing_left, None)
MyBinaryTree(3, MyBinaryTree(4), MyBinaryTree(5))
```

More implementation

```
def insert_left(self, value):
   self.left = MyBinaryTree(value, left=self.left) # right?

def insert_right(self, value):
   self.right = MyBinaryTree(value, right=self.right) # left?
```

```
def get_left_subtree(self):
   return self.left

def get_right_subtree(self):
   return self.right

def set_value(self, new_value):
   self.data = new_value

def get_value(self):
   return self.data
```