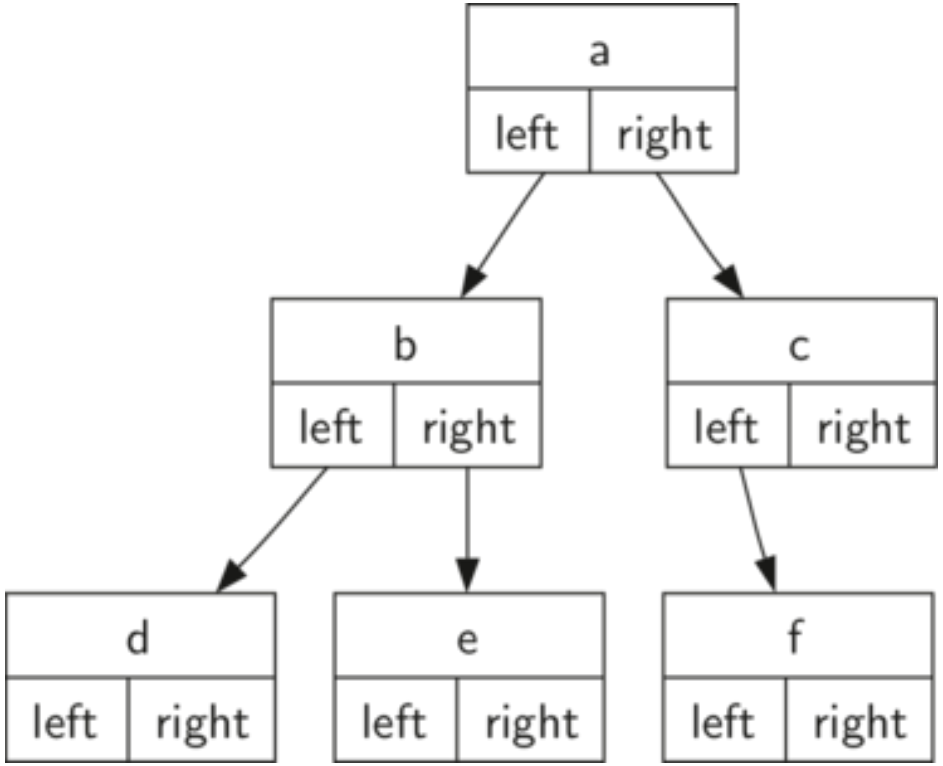
ปมและการอ้างอิง
(nodes and references)

root และ subtree ซ้าย ขวา




```
class BinaryTree:
    def __init__(self,rootObj):
        self.key = rootObj
        self.leftChild = None
        self.rightChild = None
```

insertLeft และ insertRight

```
def insertLeft(self,newNode):
    if self.leftChild == None:
        self.leftChild = BinaryTree(newNode)
    else:
        t = BinaryTree(newNode)
        t.leftChild = self.leftChild
        self.leftChild = t
```

```
def insertRight(self,newNode):
    if self.rightChild == None:
        self.rightChild = BinaryTree(newNode)
    else:
        t = BinaryTree(newNode)
        t.rightChild = self.rightChild
        self.rightChild = t
```

อธิบายเป็นภาพการทำงานได้อย่างไร?

method อื่นๆ

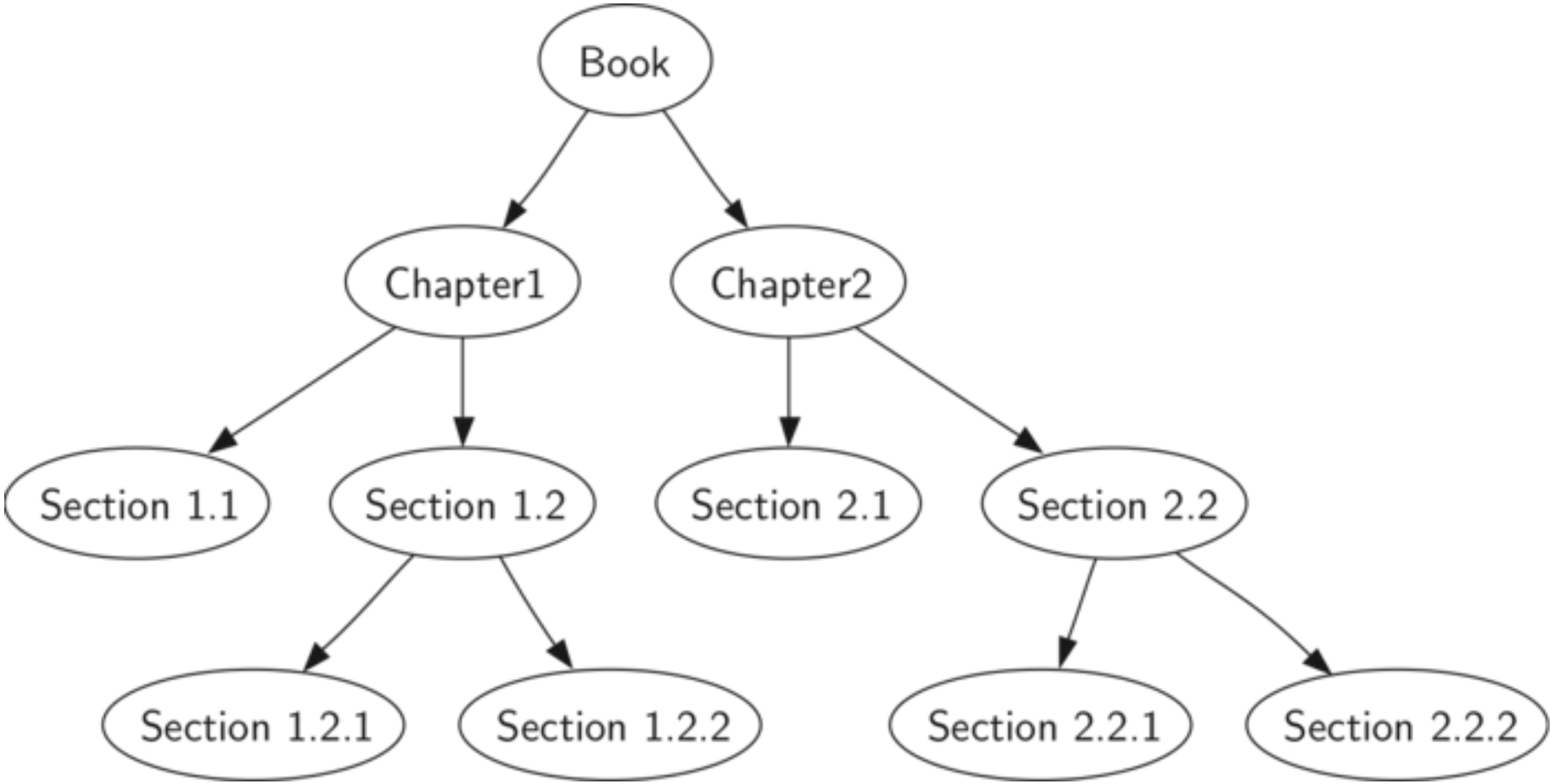
```
def getRightChild(self):
    return self.rightChild

def getLeftChild(self):
    return self.leftChild

def setRootVal(self,obj):
    self.key = obj

def getRootVal(self):
    return self.key
```

```
class BinaryTree:
   def __init__(self,rootObj):
       self.key = rootObj
       self.leftChild = None
       self.rightChild = None

  def insertLeft(self,newNode):
       if self.leftChild == None:
           self.leftChild = BinaryTree(newNode)
       else:
           t = BinaryTree(newNode)
           t.leftChild = self.leftChild
           self.leftChild = t

 def insertRight(self,newNode):
       if self.rightChild == None:
           self.rightChild = BinaryTree(newNode)
       else:
           t = BinaryTree(newNode)
           t.rightChild = self.rightChild
           self.rightChild = t


 def getRightChild(self):
       return self.rightChild

 def getLeftChild(self):
       return self.leftChild

 def setRootVal(self,obj):
       self.key = obj

 def getRootVal(self):
       return self.key


r = BinaryTree('a')
print(r.getRootVal())
print(r.getLeftChild())
r.insertLeft('b')
print(r.getLeftChild())
print(r.getLeftChild().getRootVal())
r.insertRight('c')
print(r.getRightChild())
print(r.getRightChild().getRootVal())
r.getRightChild().setRootVal('hello')
print(r.getRightChild().getRootVal())
```

การเดินทางในต้นไม้
(tree traversal)

การเดินทาง 3 แบบ

- preorder ลำดับก่อน
  - root → left subtree → right subtree
- postorder ลำดับหลัง
  - left subtree → right subtree → root
- inorder ในลำดับ
  - left subtree → root → right subtree



ควรจะเดินทางแบบไหน?

preorder

```
def preorder(tree):
    if tree:
        print(tree.getRootVal())
        preorder(tree.getLeftChild())
        preorder(tree.getRightChild())
```

แบบ function แยก

```
def preorder(self):
    print(self.key)
    if self.leftChild:
        self.leftChild.preorder()
    if self.rightChild:
        self.rightChild.preorder()
```

แบบ method

postorder

```
def postorder(tree):
    if tree != None:
        postorder(tree.getLeftChild())
        postorder(tree.getRightChild())
        print(tree.getRootVal())

def postordereval(tree):
    opers = {'+':operator.add, '-':operator.sub, '*':operator.mul, '/':operator.truediv}
    res1 = None
    res2 = None
    if tree:
        res1 = postordereval(tree.getLeftChild())
        res2 = postordereval(tree.getRightChild())
        if res1 and res2:
            return opers[tree.getRootVal()](res1,res2)
        else:
            return tree.getRootVal()
```

inorder

```
def inorder(tree):
  if tree != None:
      inorder(tree.getLeftChild())
      print(tree.getRootVal())
      inorder(tree.getRightChild())
```

```
def printexp(tree):
  sVal = ""
  if tree:
      sVal = '(' + printexp(tree.getLeftChild())
      sVal = sVal + str(tree.getRootVal())
      sVal = sVal + printexp(tree.getRightChild())+')'
  return sVal
```