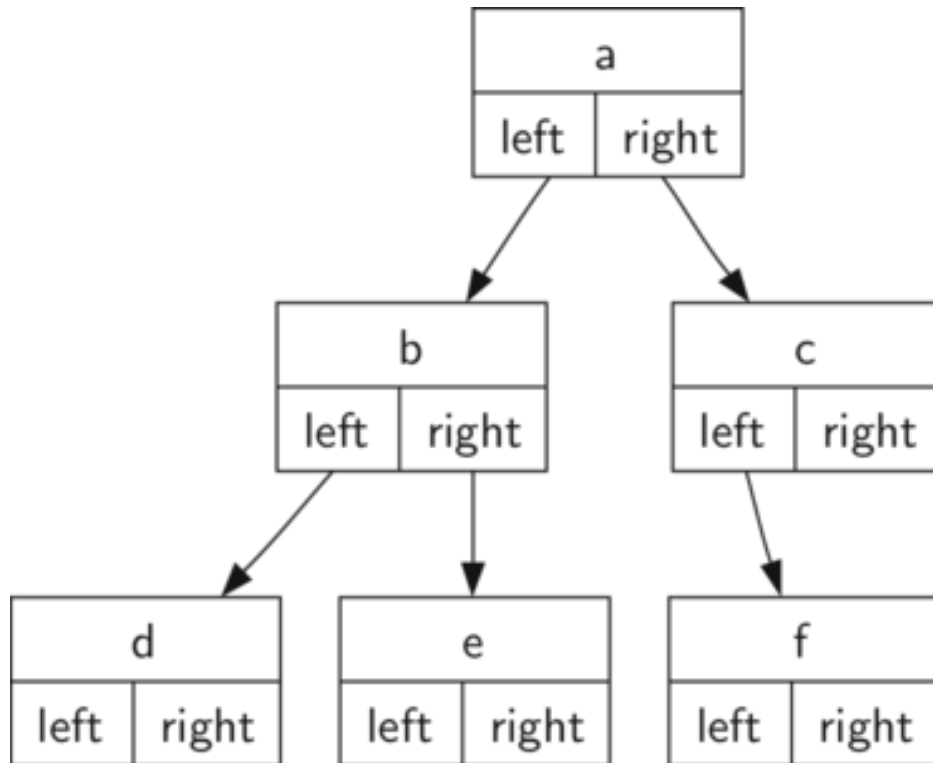


ปมและการอ้างอิง
(nodes and references)

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
```

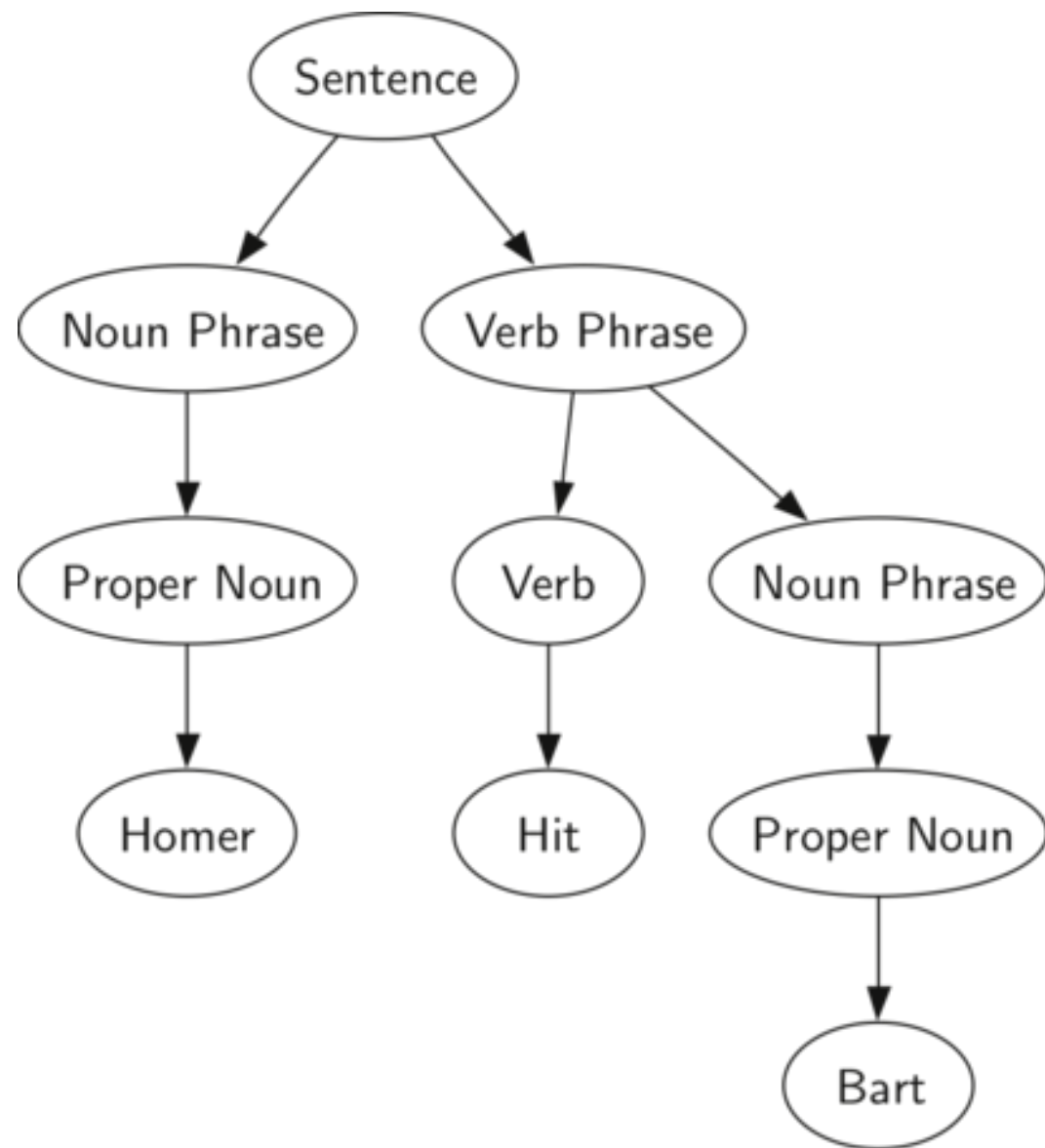
```
def setRootVal(self,obj):  
    self.key = obj
```

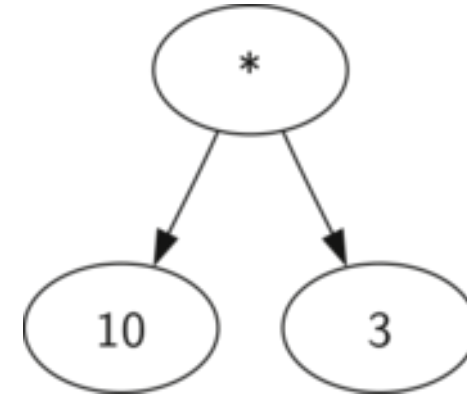
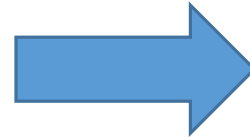
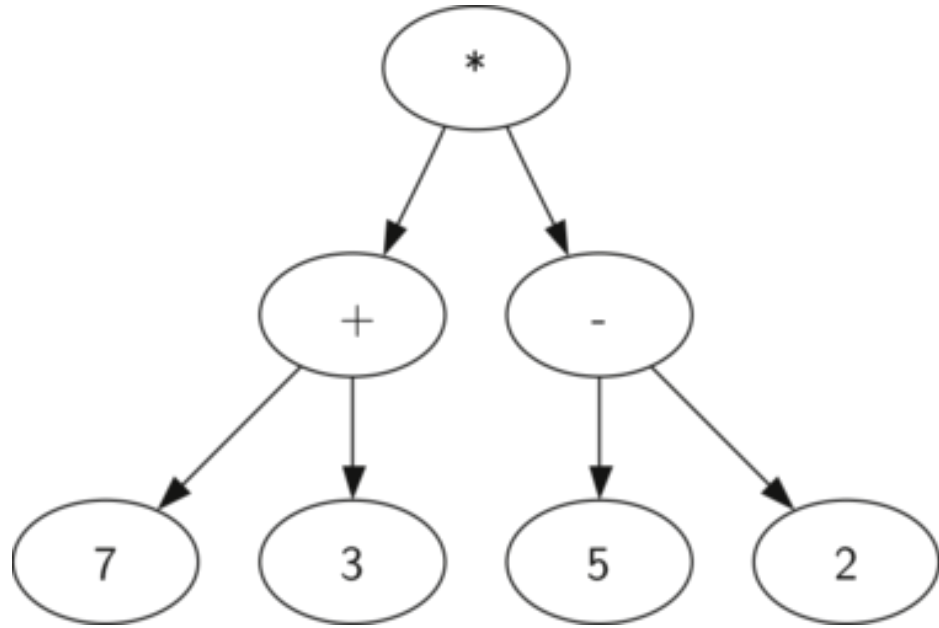
```
def getRootVal(self):  
    return self.key
```

```
1 class BinaryTree:
2     def __init__(self, rootObj):
3         self.key = rootObj
4         self.leftChild = None
5         self.rightChild = None
6
7     def insertLeft(self, newNode):
8         if self.leftChild == None:
9             self.leftChild = BinaryTree(newNode)
10        else:
11            t = BinaryTree(newNode)
12            t.leftChild = self.leftChild
13            self.leftChild = t
14
15    def insertRight(self, newNode):
16        if self.rightChild == None:
17            self.rightChild = BinaryTree(newNode)
18        else:
19            t = BinaryTree(newNode)
20            t.rightChild = self.rightChild
21            self.rightChild = t
22
23
```

```
24 def getRightChild(self):
25     return self.rightChild
26
27 def getLeftChild(self):
28     return self.leftChild
29
30 def setRootVal(self, obj):
31     self.key = obj
32
33 def getRootVal(self):
34     return self.key
35
36
37 r = BinaryTree('a')
38 print(r.getRootVal())
39 print(r.getLeftChild())
40 r.insertLeft('b')
41 print(r.getLeftChild())
42 print(r.getLeftChild().getRootVal())
43 r.insertRight('c')
44 print(r.getRightChild())
45 print(r.getRightChild().getRootVal())
46 r.getRightChild().setRootVal('hello')
47 print(r.getRightChild().getRootVal())
48
```

ต้นไม้วิเคราะห์คำ
(parse tree)





$$(7 + 3) * (5 - 2) \rightarrow 10 * 3$$

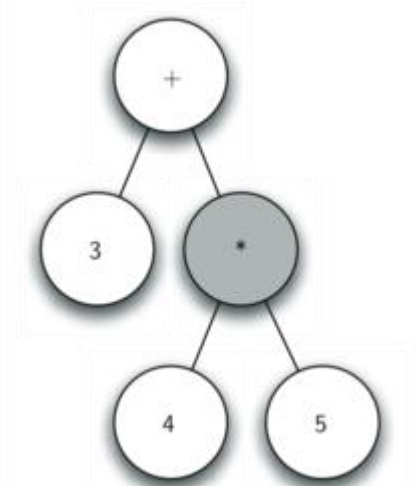
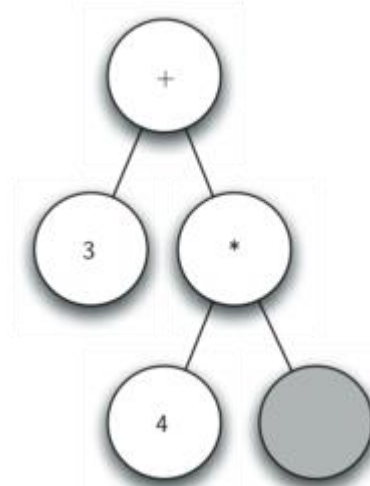
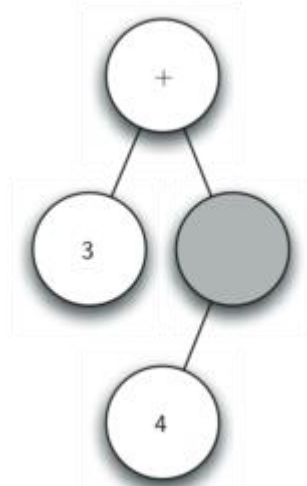
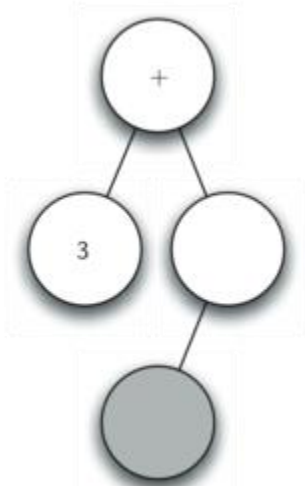
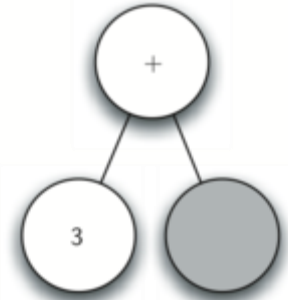
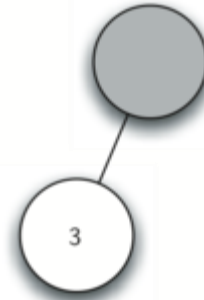
เราจะเรียนรู้อะไรบ้างจากต้นไม้วิเคราะห์ค่า

- สร้างต้นไม้วิเคราะห์ค่าอย่างไร
- ประมวลผลค่าที่ถูกแทนด้วยต้นไม้วิเคราะห์ค่าได้อย่างไร
- คำนวณประโยคที่ถูกแทนด้วยต้นไม้วิเคราะห์ค่าได้อย่างไร

แยกประเภทของกลุ่มคำ

- วงเล็บเปิด (
 - เพิ่ม left child node
 - เลื่อนจุดสนใจไปยัง left child
- +, -, *, /
 - กำหนดค่าเครื่องหมายให้กับ node (root ของ subtree)
 - เพิ่ม right child node
 - เลื่อนจุดสนใจไปยัง right child
- ตัวเลข
 - กำหนดค่าตัวเลขให้กับ node
 - เลื่อนจุดสนใจไปยัง parent
- วงเล็บปิด)
 - เลื่อนจุดสนใจไปยัง parent

$(3 + (4 * 5))$



tree และ stack

- มีการติดตามจุดสนใจ
- tree
 - getLeftChild
 - getRightChild
- stack เก็บ parent node
 - ทุกครั้งที่มีการเลื่อนไป child node ให้ push node ที่เป็นจุดสนใจปัจจุบันลงไป
 - ทุกครั้งที่มีการกลับไปหา parent node ให้ pop ออกจะได้ parent node

```
1from pythonds.basic.stack import Stack
2from pythonds.trees.binaryTree import BinaryTree
3
4def buildParseTree(fpexp):
5    fplist = fpexp.split()
6    pStack = Stack()
7    eTree = BinaryTree('')
8    pStack.push(eTree)
9    currentTree = eTree
10    for i in fplist:
11        if i == '(':
12            currentTree.insertLeft('')
13            pStack.push(currentTree)
14            currentTree = currentTree.getLeftChild()
15        elif i not in ['+', '-', '*', '/', ')']:
```

```
16            currentTree.setRootVal(int(i))
17            parent = pStack.pop()
18            currentTree = parent
19        elif i in ['+', '-', '*', '/']:
20            currentTree.setRootVal(i)
21            currentTree.insertRight('')
22            pStack.push(currentTree)
23            currentTree = currentTree.getRightChild()
24        elif i == ')':
25            currentTree = pStack.pop()
26        else:
27            raise ValueError
28    return eTree
29
30pt = buildParseTree("( ( 10 + 5 ) * 3 )")
31pt.postorder
32
```

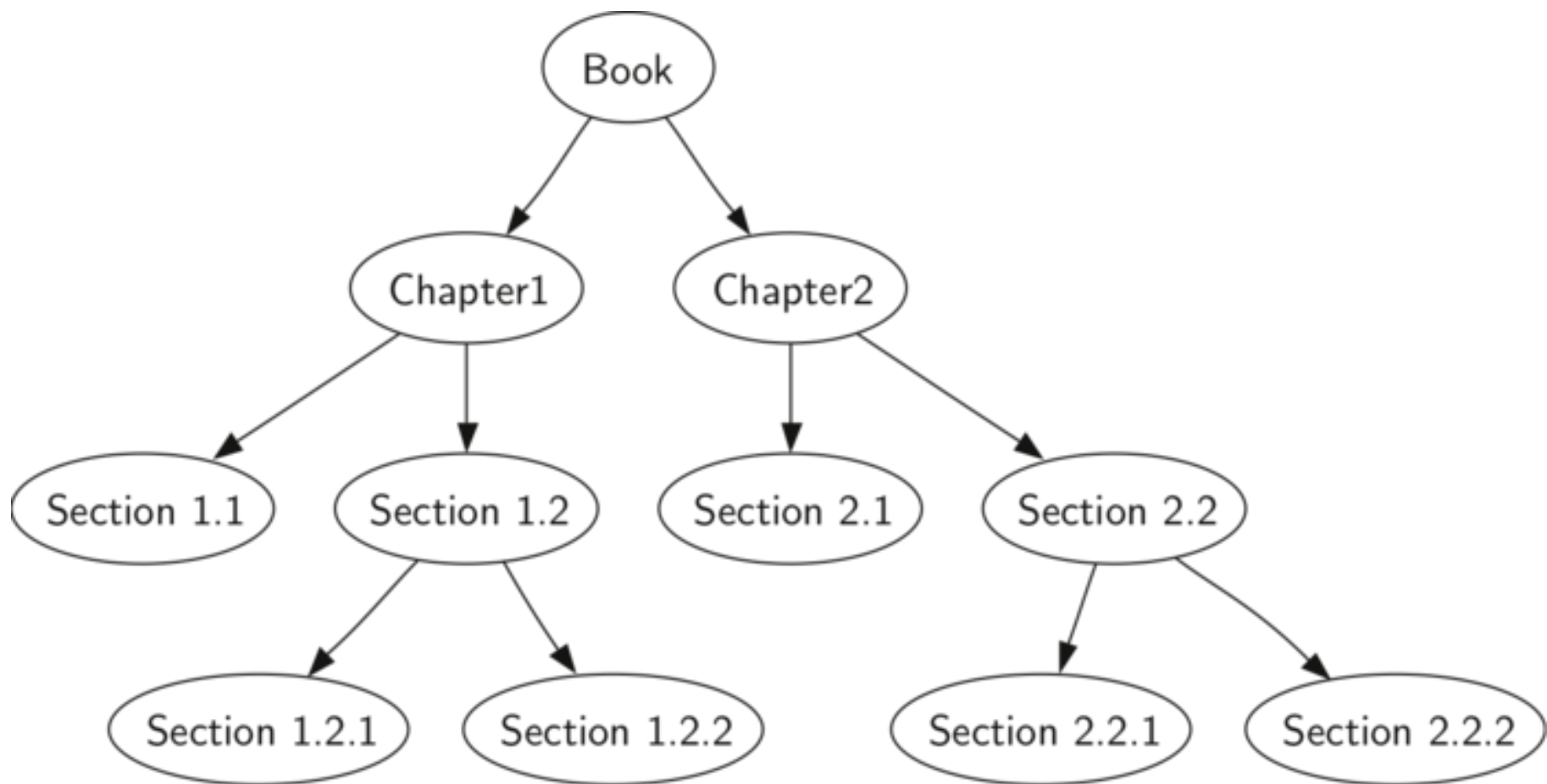
การประมวลผลค่า

```
def evaluate(parseTree):  
    ops = {'+':operator.add, '-':operator.sub, '*':operator.mul, '/':operator.truediv}  
  
    leftC = parseTree.getLeftChild()  
    rightC = parseTree.getRightChild()  
  
    if leftC and rightC:  
        fn = ops[parseTree.getRootVal()]  
        return fn(evaluate(leftC),evaluate(rightC))  
    else:  
        return parseTree.getRootVal()
```

การเดินทางในต้นไม้ (tree traversal)

การเดินทาง 3 แบบ

- preorder ลำดับก่อน
 - root \rightarrow left subtree \rightarrow right subtree
- postorder ลำดับหลัง
 - left subtree \rightarrow right subtree \rightarrow root
- inorder ในลำดับ
 - left subtree \rightarrow root \rightarrow 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

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

postorder

```
def postordereval(tree):  
    ops = {'+':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 ops[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
```