Ceng 111 – Fall 2020
Week 13

Complexity, ADT

Credit: Some slides are from the “Invitation to Computer Science” book by G. M. Schneider, J. L. Gersting and some from the “Digital Design” book by M. M. Mano and M. D. Ciletti.

Today

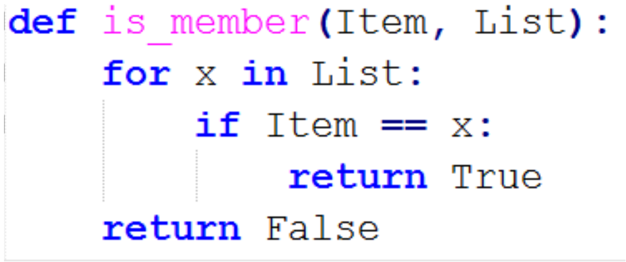
- Finalize complexity with a few examples
- ADT
  - Stack
  - Queue
  - Tree

Administrative Notes

METU Computer Engineering

- Live sessions
  - Tue 13:40 Session
  - Wed 10:40 Session
- Social session
- The labs
- Office hours: Tue 10:30
- Lab Exam 3: 16 January
- Final: 30 January 13:30

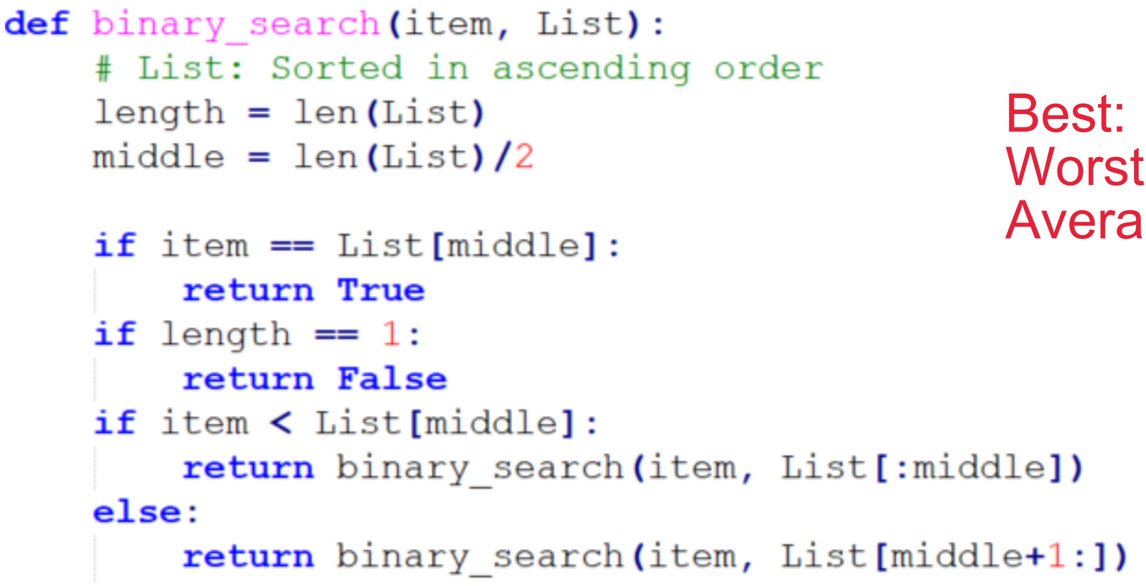
Complexity Analysis Examples



```
def is_member(Item, List):
    for x in List:
        if Item == x:
            return True
    return False
```

	Complexity
Best:	O(1)
Worst:	O(n)
Average:	O(n)

Complexity Analysis Examples



```
def binary_search(item, List):
    # List: Sorted in ascending order
    length = len(List)
    middle = len(List)/2

    if item == List[middle]:
        return True
    if length == 1:
        return False
    if item < List[middle]:
        return binary_search(item, List[:middle])
    else:
        return binary_search(item, List[middle+1:])
```

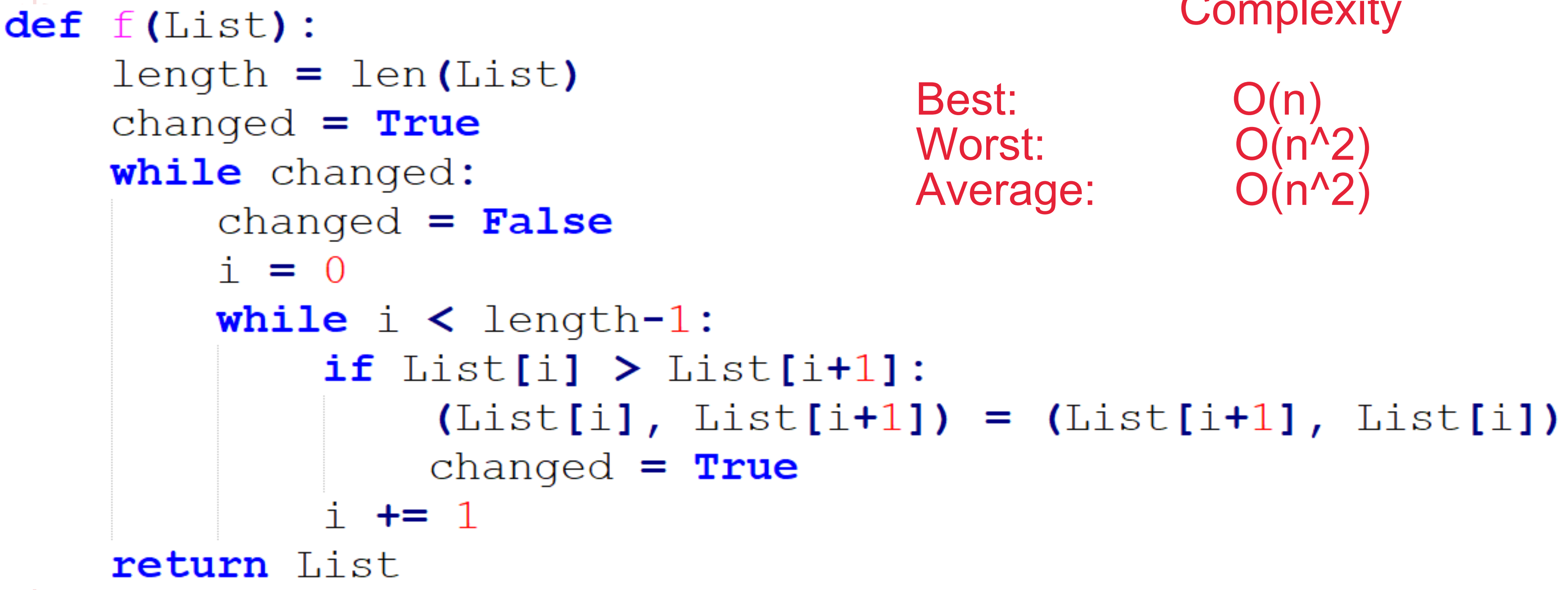
	Complexity
Best:	O(1)
Worst:	O(logn)
Average:	O(logn)

Complexity Analysis Examples

```
def csort(A):
        # Assume that the numbers are in the range 1,...,k
        k = max(A)
        C = [0] * k

        # Count the numbers in A
        for x in A:
                C[x-1] += 1

        # Accumulate the counts in C
        i = 1
        while i < k:
                C[i] += C[i-1] i += 1

        # Place the numbers into correct locations
        B = [0] * len(A)
        for x in A:
                B[C[x-1]-1] = x C[x-1] -= 1

        return B
```

	Complexity
Best:	O(n+k)
Worst:	O(n+k)
Average:	O(n+k)

Complexity Analysis Examples



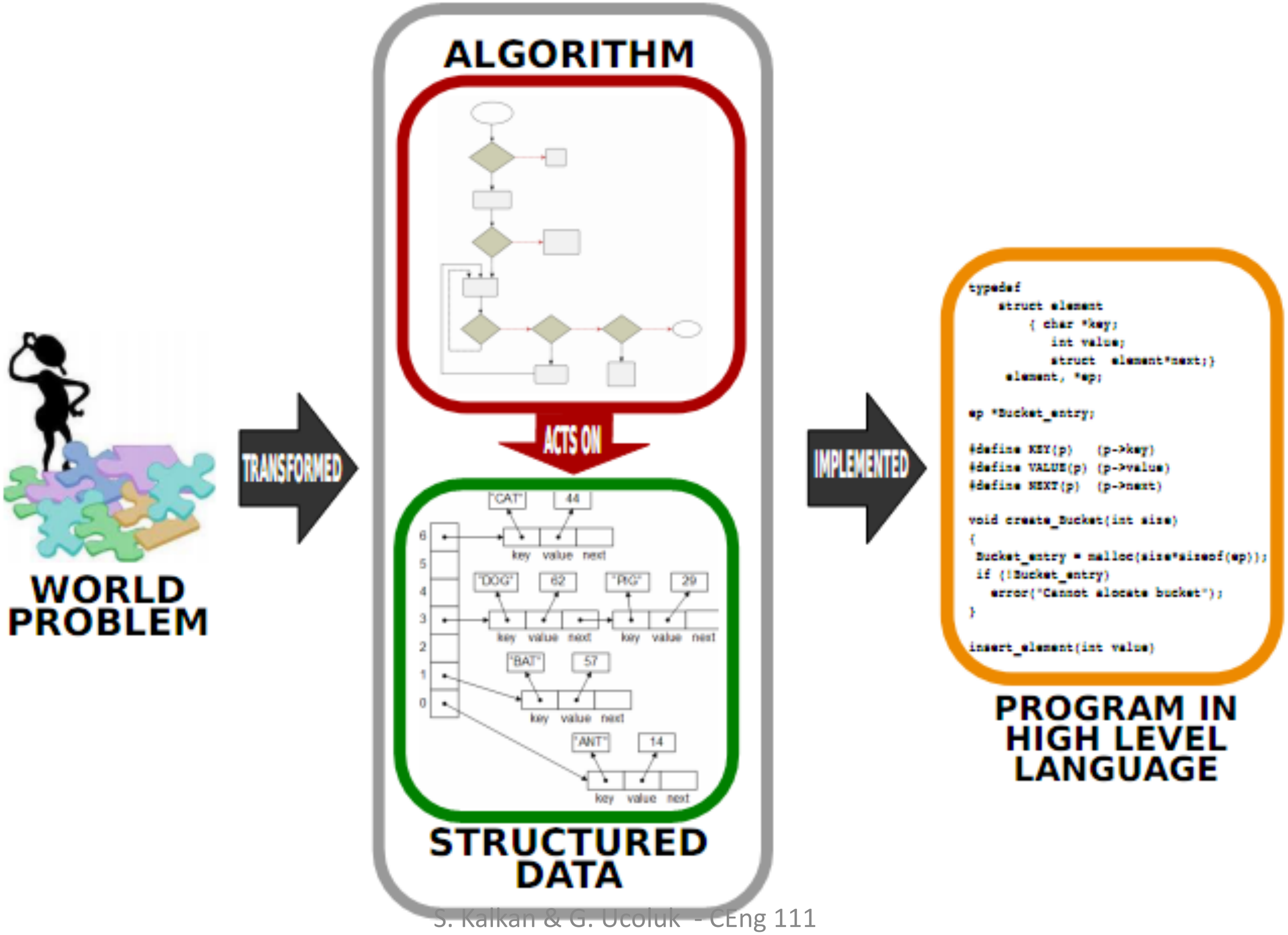
```
def f(List):
    length = len(List)
    changed = True
    while changed:
        changed = False
        i = 0
        while i < length-1:
            if List[i] > List[i+1]:
                (List[i], List[i+1]) = (List[i+1], List[i])
                changed = True
            i += 1
    return List
```

	Complexity
Best:	O(n)
Worst:	O(n^2)
Average:	O(n^2)

Exercises on Complexity

1. What is the complexity of the following?
   - Finding the minimum or the maximum in a list, which is (a) sorted or (b) unsorted.
   - Finding the average of numbers in a list.
2. Assume that we have a sorted list L.
   - What is the complexity of sorting L after inserting a new number?
3. What is the complexity of checking whether a list is sorted?

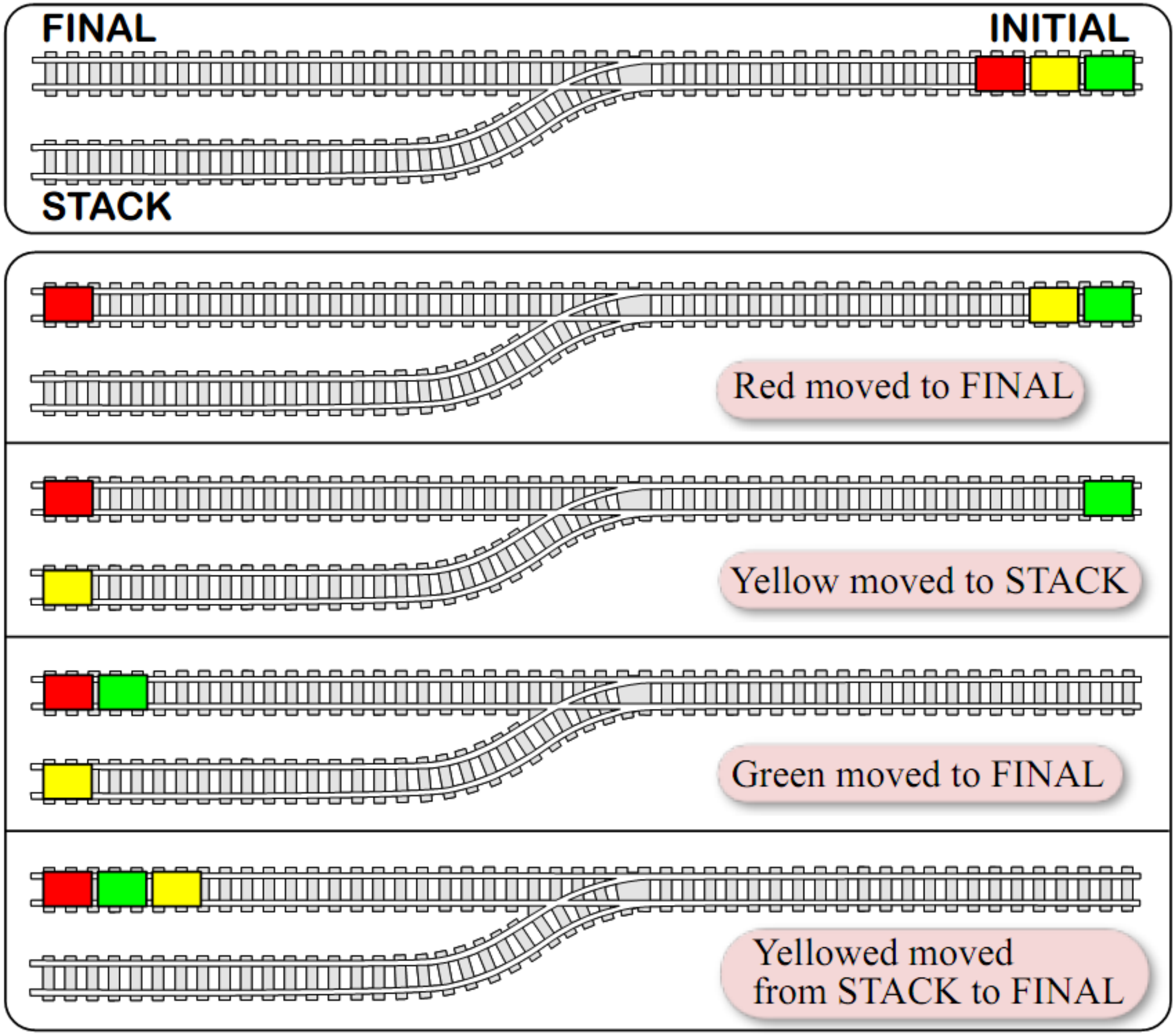
Design of a solution



```
typedef
    struct element
        { char *key;
           int value;
           struct  element*next;}
     element, *ep;

ep *Bucket_entry;

#define KEY(p)   (p->key)
#define VALUE(p) (p->value)
#define NEXT(p)  (p->next)

void create_Bucket(int size)
{
 Bucket_entry = malloc(size*sizeof(ep));
 if (!Bucket_entry)
   error("Cannot alocate bucket");
}

insert_element(int value)
```

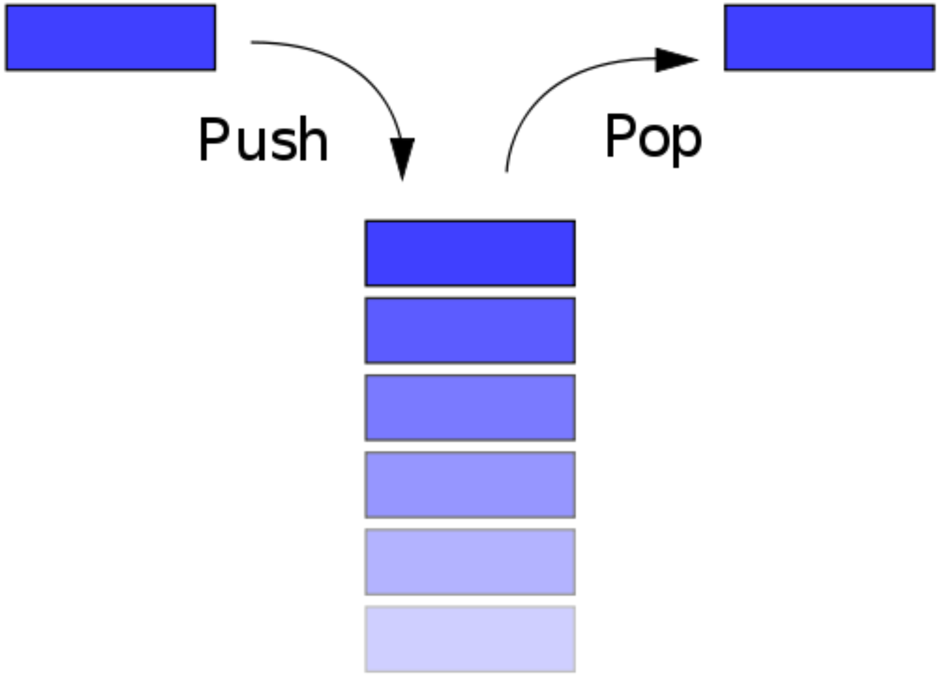
ABSTRACT DATA TYPES

Remember Stacks?



Stacks

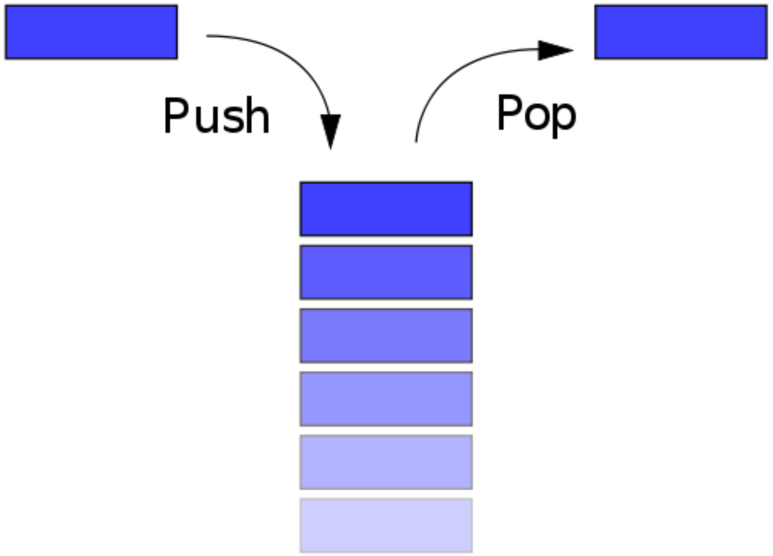
- LIFO:
  - Last In First Out
- We have seen it before (in the Shunting-Yard algorithm)
- Main operations:
  - Push
  - Pop



METU Computer Engineering

Stacks (cont'd)

- Operations:
  1. Push
  2. Pop
  3. Top/Peek
     - Get the top element without removing it
  4. Is-Empty
     - Checks whether the stack is empty
  5. Length
     - # of elements



Stacks in Python

Stack Operation	Corresponding Python Op.
Pop	L.pop()
Push	L.append(item)
Top/Peek	L[-1]
Is-Empty	L == []
Length	len(L)

Implementing Stacks in Python

```
def CreateStack():
     """Creates an empty stack"""
     return []

def Push(item, Stack):
     """Add item to the top of Stack"""
     Stack.append(item)

def Pop(Stack):
     """Remove and return the item at the top of the Stack"""
     return Stack.pop()

def Top(Stack):
     """Return the value of the item at the top of the
          Stack without removing it"""
     return Stack[-1]

def IsEmpty(Stack):
     """Check whether the Stack is empty"""
     return Stack == []
```

Stacks in Python (Example)

- Implement postfix implementation in Python using stacks.
  - Given a string like “3 4 + 5 7 + *”, evaluate and return the result.

Stacks in Python (Example - Solution)

```
def postfix_eval(Exp):
        # Example Exp: “3 4 + 5 6 + *”
        Stack = CreateStack()
        Exp = Exp.split(’ ’)

        for token in Exp:
                if token.isdigit(): Push(token, Stack)
                else:
                        op2 = Pop(Stack)
                        op1 = Pop(Stack)
                        result = str(eval(op1 + token + op2))
                        Push(result, Stack)

        return Pop(Stack)
```

METU Computer Engineering

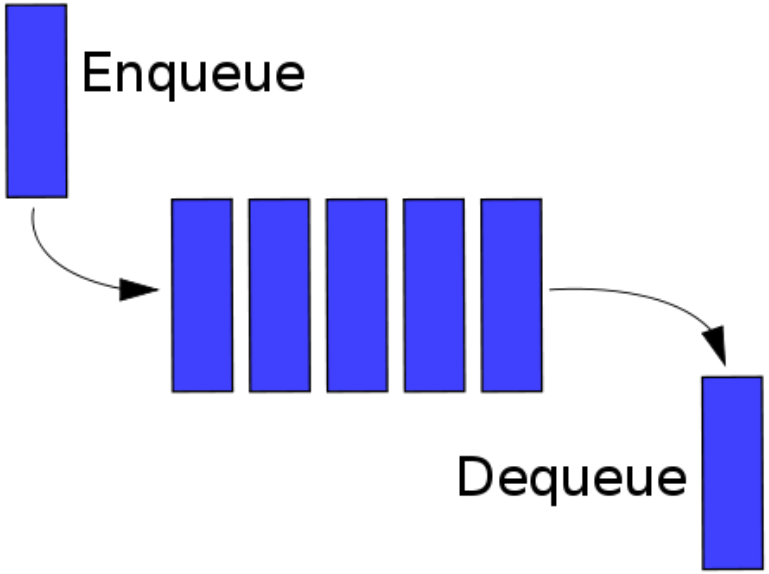
Stacks: Formal definition

$push(item, stack)$ → $item \odot stack$

- $new() \rightarrow \emptyset$
- $popoff(\xi \odot S) \rightarrow S$
- $top(\xi \odot S) \rightarrow \xi$
- $isempty(\emptyset) \rightarrow$TRUE
- $isempty(\xi \odot S) \rightarrow$ FALSE

Queues

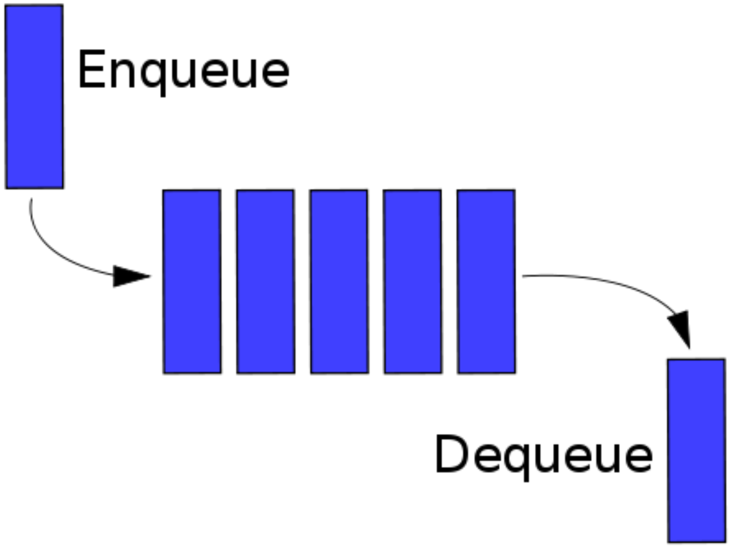
- FIFO:
  - First In First Out
- The item that was inserted first is removed first.
- Main operations:
  - Add
  - Remove



METU Computer Engineering

Queues (cont'd)

- Operations:
  1. Add
  2. Remove
  3. Front/Peek
  4. Is-Empty
  5. Length



Queues in Python

Queue Operation	Corresponding Python Op.
Add	L.append(item)
Remove	L.pop(0)
Front/Peek	L[0]
Is-Empty	L == []
Length	len(L)

METU Computer Engineering

Implementing Queues in Python

```
def CreateQueue():
        """Creates an empty queue"""
        return []

def Enqueue(item, Queue):
        """Add item to the end of Queue"""
        Queue.append(item)

def Dequeue(Queue):
        """Remove and return the item at the front of the Queue"""
        return Queue.pop(0)

def IsEmpty(Queue):
        """Check whether the Queue is empty"""
        return Queue == []

def Front(Queue):
        """Return the value of the current front item without removing it"""
        return Queue[0]
```

Queues: Formal Definition

$$add(item, queue) \longrightarrow item \boxplus queue$$

- $new() \rightarrow \varnothing$
- $front(\xi \boxplus \varnothing) \rightarrow \xi$
- $front(\xi \boxplus Q) \rightarrow front(Q)$
- $remove(\xi \boxplus \varnothing) \rightarrow \varnothing$
- $remove(\xi \boxplus Q) \rightarrow \xi \boxplus remove(Q)$
- $isempty(\varnothing) \rightarrow$TRUE
- $isempty(\xi \boxplus Q) \rightarrow$ FALSE

Queues in Python (Example)

These two functions should run on two “threads” that share some memory together

```
CustomerQueue = CreateQueue()

def bank_queue():
        while True:
                if NewCustomerArrived() == True:
                        new_customer = GetCustomerInfo()
                        Enqueue(new_customer, CustomerQueue)

def serve_customers():
        while True:
                if CustomerQueue.IsEmpty() == False:
                        customer = Dequeue(CustomerQueue)
                        ServeCustomer(customer)
```

Priority Queue

- Similar to Queue except that the items in a queue has a priority value based on which they are kept in order!
- Operations:
  - insert(item, priority) → Push item with the given priority
  - Highest() → The item in the queue that has the highest priority
  - Deletehighest() → Delete the item that has the highest priority
  - Is-Empty
  - Length

Priority Queues in Python

Priority Queue Operation	**Corresponding Python Op.**
Insert	L.append((item, priority))
Highest	Write a function that finds the max
Delete highest	Write a function that finds the max and deletes it
Is-Empty	L == []
Length	len(L)

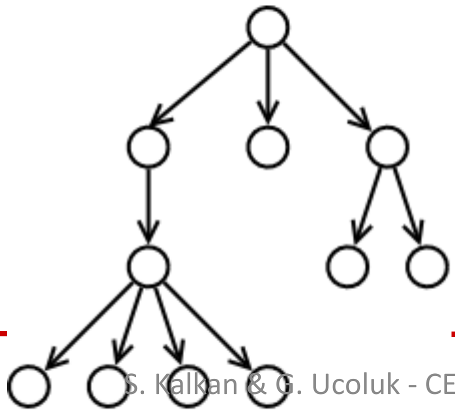
$insert(item, PQ) \Rightarrow item \curvearrowright PQ$

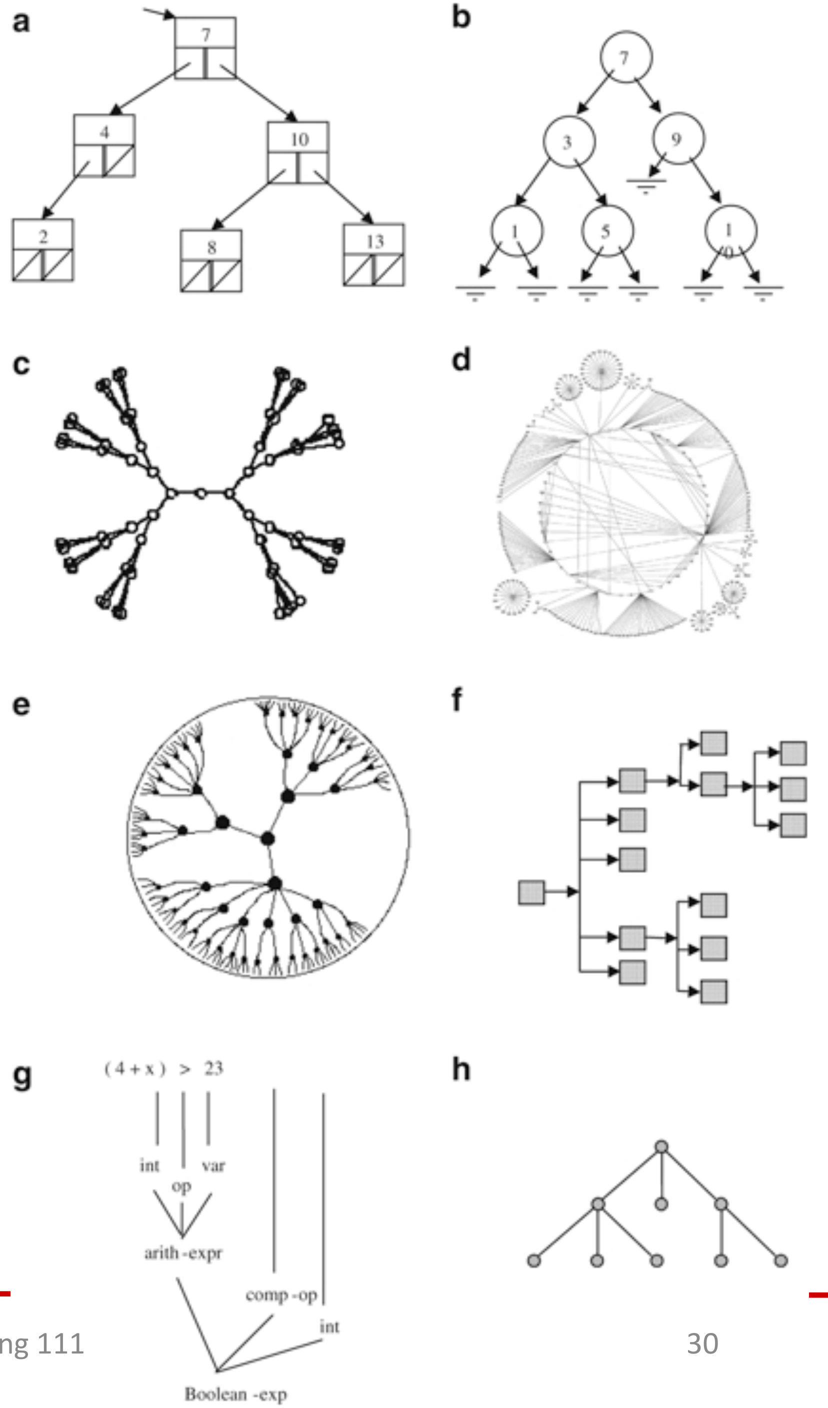
- $new() \rightarrow \between$
- $highest(\xi \curvearrowright \between) \rightarrow \xi$
- $highest(\xi \curvearrowright PQ) \rightarrow$
  if $priority(\xi) > priority(highest(PQ))$ then ξ else $highest(PQ)$
- $deletehighest(\xi \curvearrowright \between) \rightarrow \between$
- $deletehighest(\xi \curvearrowright PQ) \rightarrow$
  if $priority(\xi) > priority(highest(PQ))$ then PQ
  else $\xi \curvearrowright deletehighest(PQ)$
- $isempty(\between) \rightarrow$TRUE
- $isempty(\xi \curvearrowright PQ) \rightarrow$ FALSE

METU Computer Engineering

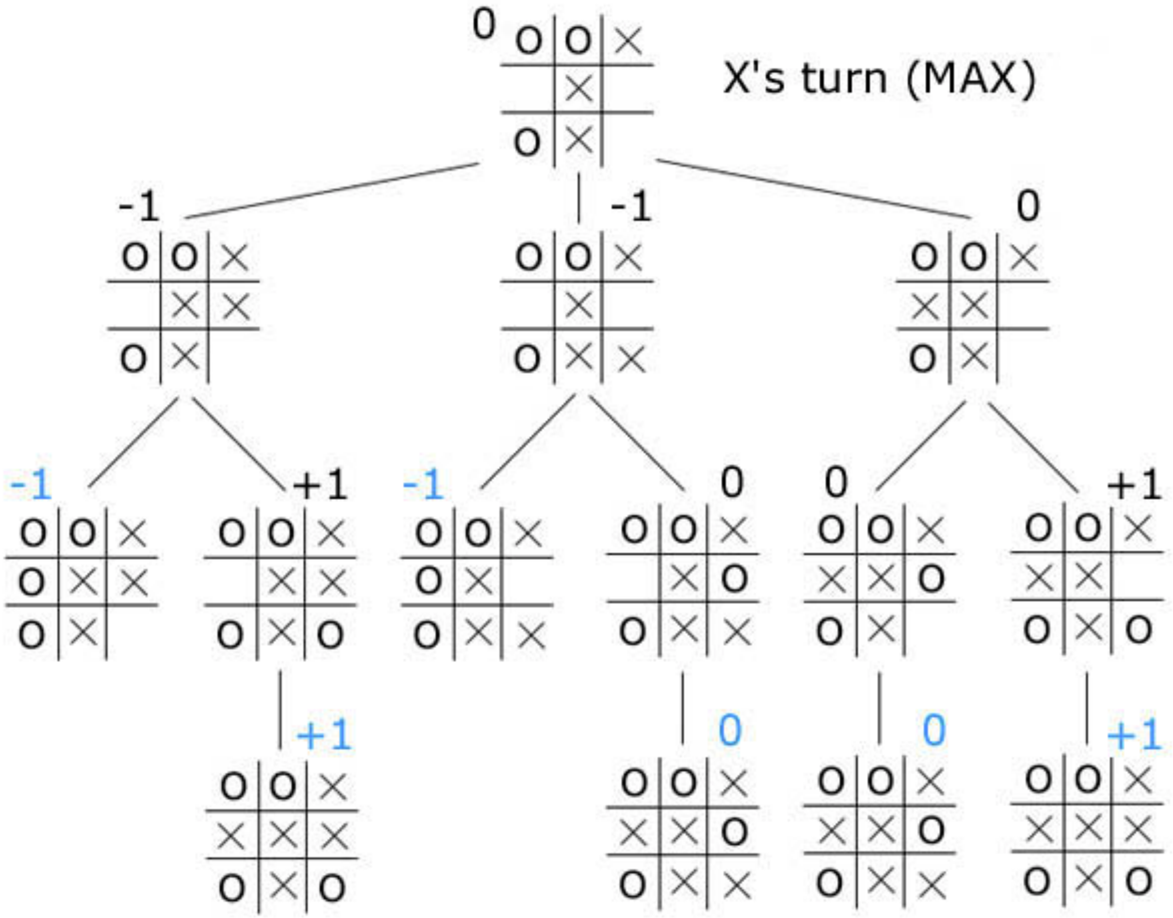
METU Computer Engineering

Trees



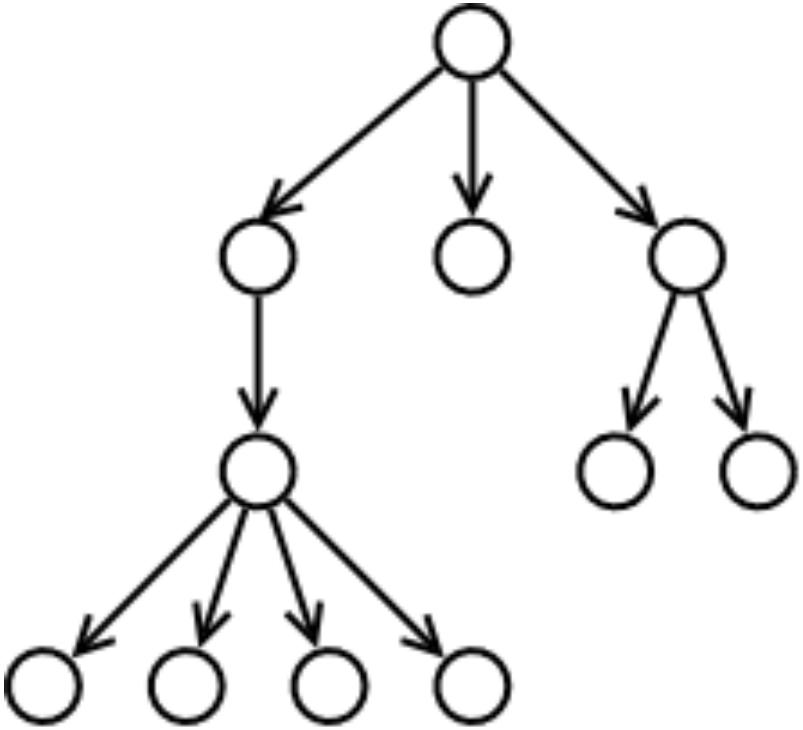


Example for Trees: Decision/Game Tree



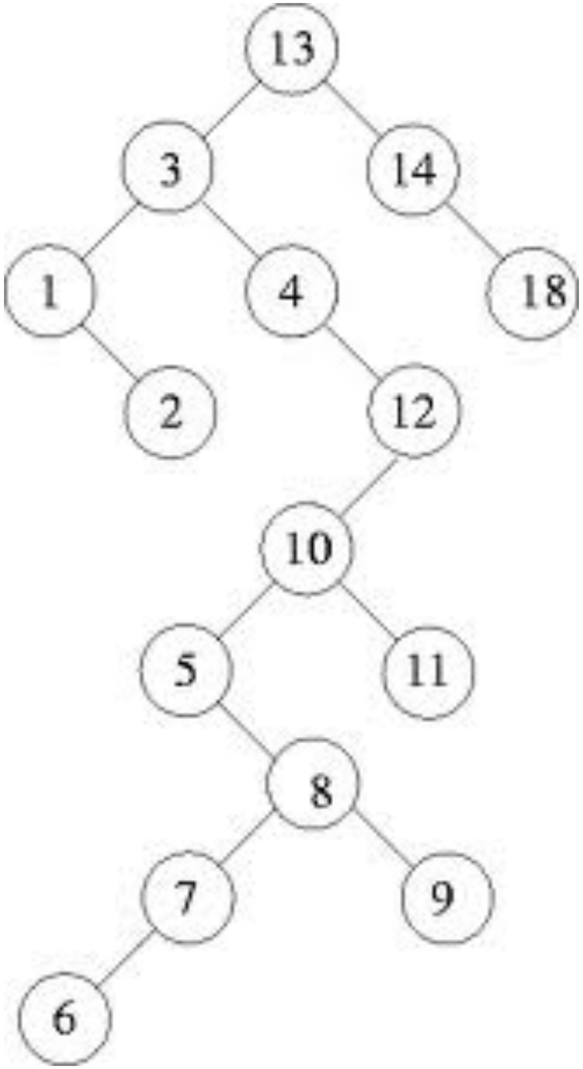
Properties of Trees

- A tree is composed of nodes.
- A node can have either no branches, two branches or more than two branches.
- Binary tree: a tree where nodes have two branches.
- The depth of a tree:
  - The number of levels in the tree.



Binary Search Tree

- The nodes in the left branch of a node have less value than the node.
- The nodes in the right branch of a node have more value than the node.



How can we represent Trees in Python?

- Nested Lists

vs.

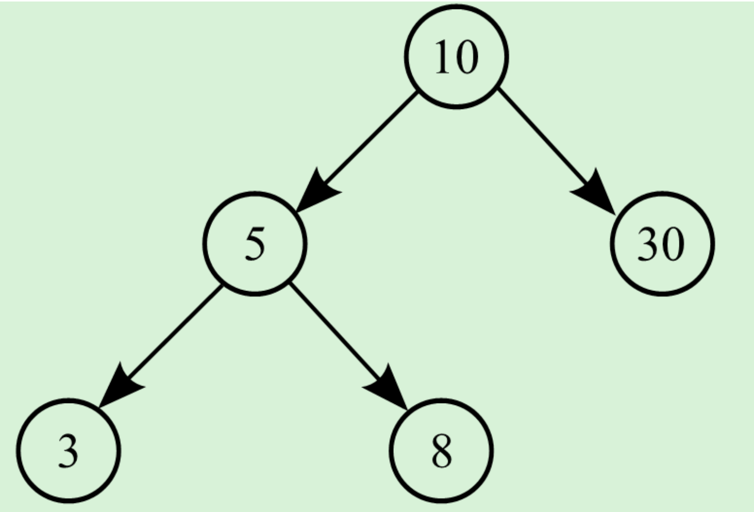
- Nested Tuples

- Tuples / Lists

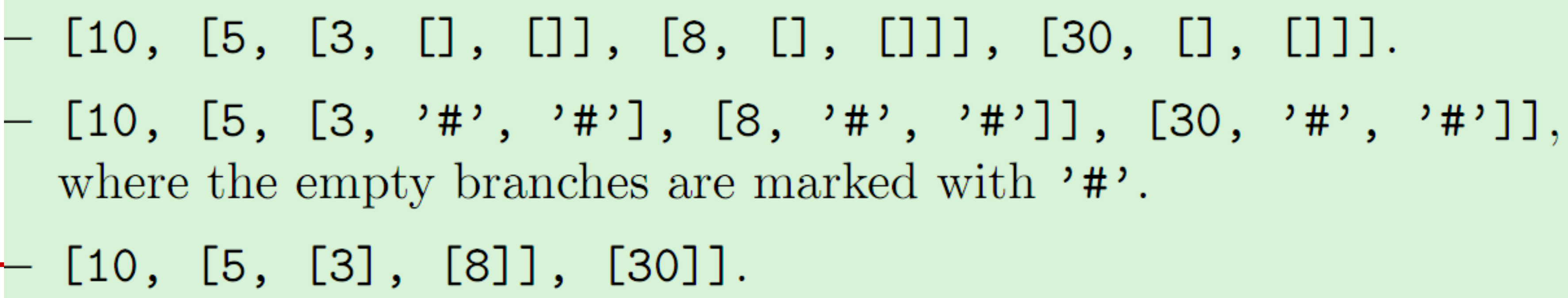
vs.

- Dictionaries

Now, let us see how we can represent Trees in Python



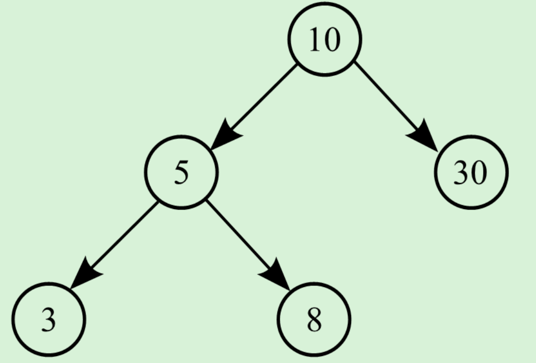
- Using Lists:

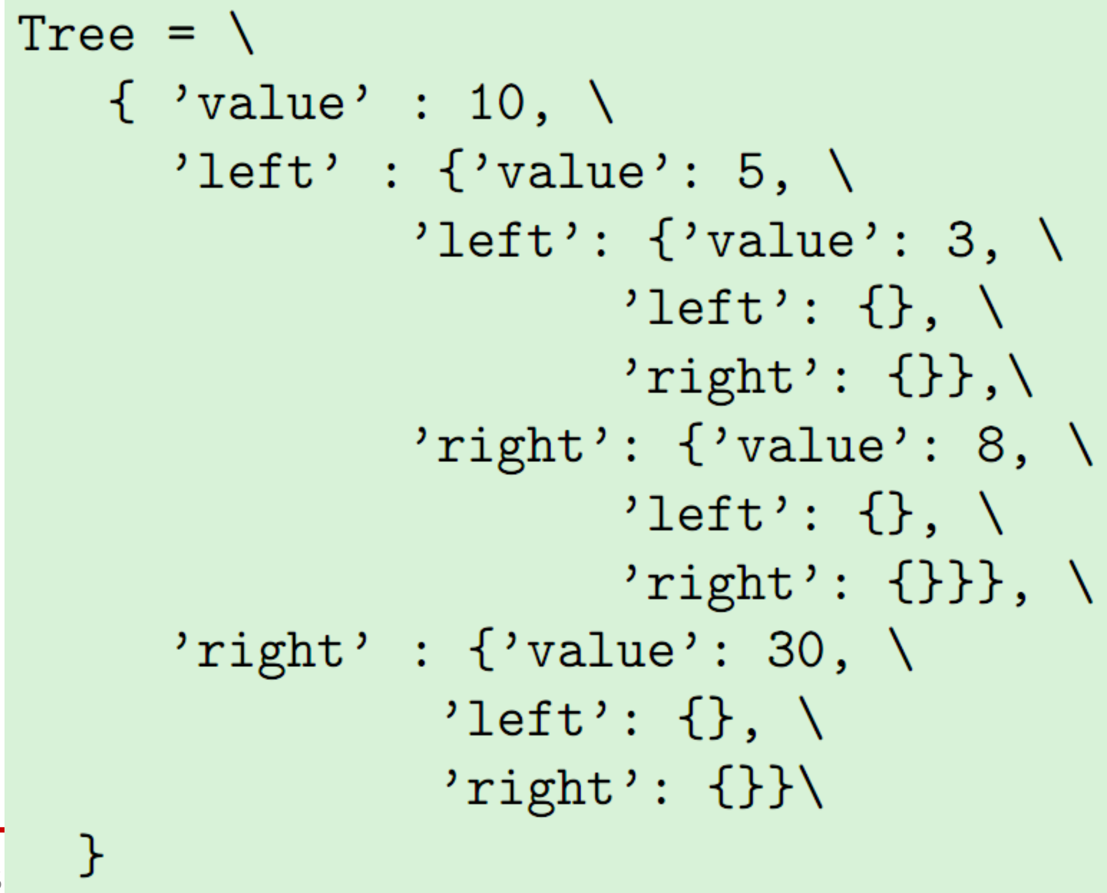


- `[10, [5, [3, [], []], [8, [], []]], [30, [], []]]`.
- `[10, [5, [3, ’#’, ’#’], [8, ’#’, ’#’]], [30, ’#’, ’#’]]`, where the empty branches are marked with `’#’`.
- `[10, [5, [3], [8]], [30]]`.

Now, let us see how we can represent Trees in Python

- Using dictionaries

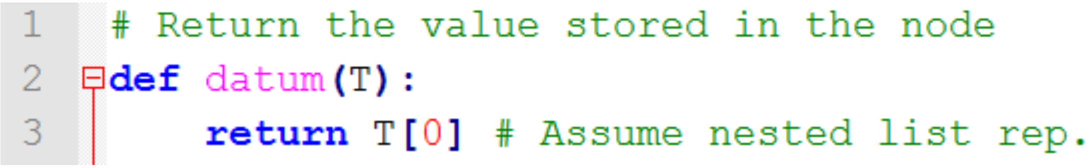




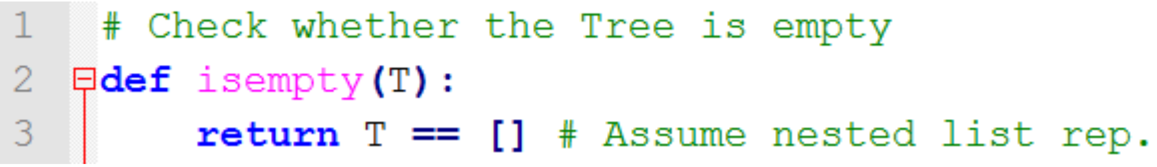
```
Tree = \
    { ’value’ : 10, \
      ’left’ : {’value’: 5, \
                ’left’: {’value’: 3, \
                         ’left’: {}, \
                         ’right’: {}},\
                ’right’: {’value’: 8, \
                          ’left’: {}, \
                          ’right’: {}}}, \
      ’right’ : {’value’: 30, \
                 ’left’: {}, \
                 ’right’: {}}\
  }
```

Tree operations

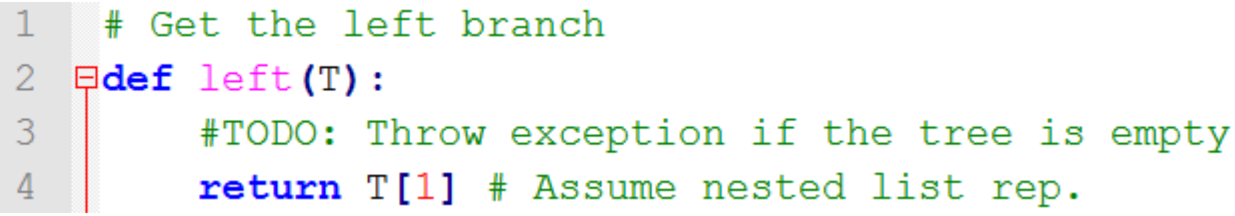
- datum()
- isempty()
- left()
- right()
- createNode()



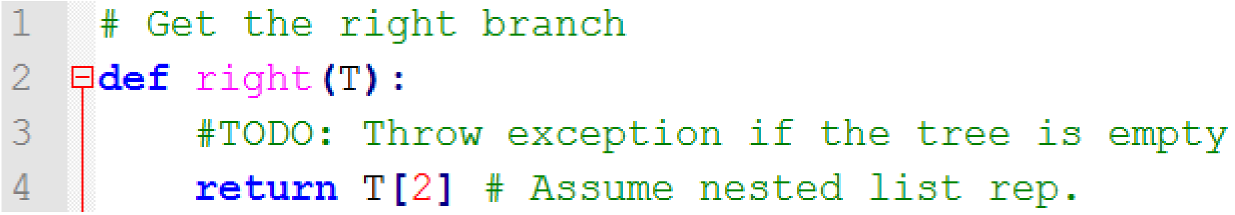
```python
# Return the value stored in the node
def datum(T):
    return T[0] # Assume nested list rep.
```



```python
# Check whether the Tree is empty
def isempty(T):
    return T == [] # Assume nested list rep.
```

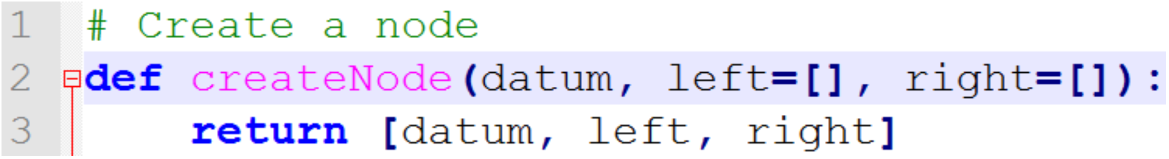


```python
# Get the left branch
def left(T):
    #TODO: Throw exception if the tree is empty
    return T[1] # Assume nested list rep.
```



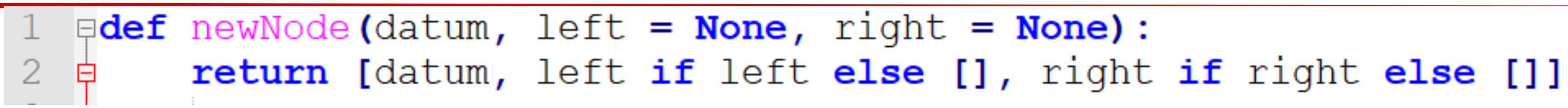
```python
# Get the right branch
def right(T):
    #TODO: Throw exception if the tree is empty
    return T[2] # Assume nested list rep.
```

This creates aliasing ➔



```python
# Create a node
def createNode(datum, left=[], right=[]):
    return [datum, left, right]
```

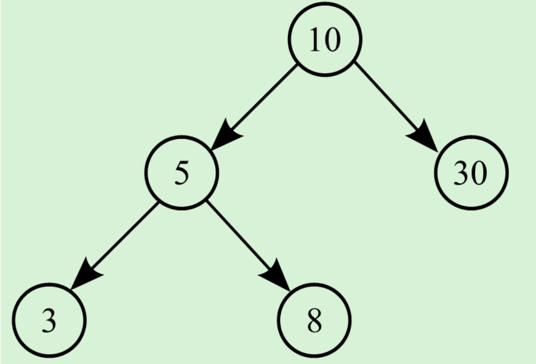
Use the following:



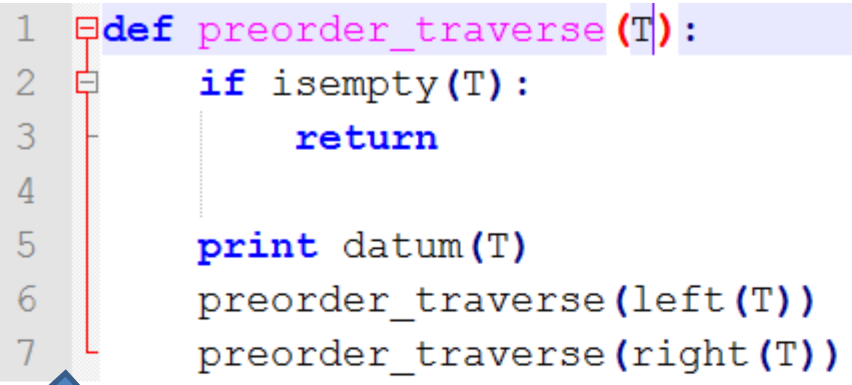
```python
def newNode(datum, left = None, right = None):
    return [datum, left if left else [], right if right else []]
```

Traversing Trees

1. Pre-order Traversal



```
[10, [5, [3, [], []], [8, [], []]], [30, [], []]].
```

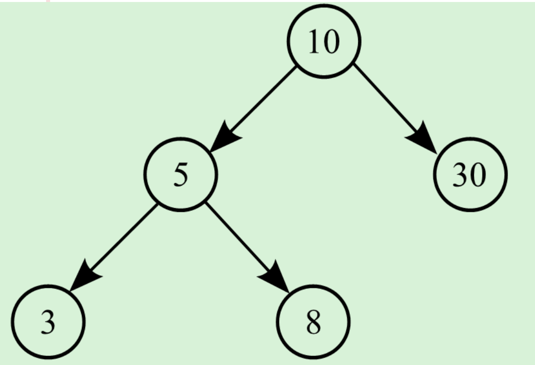


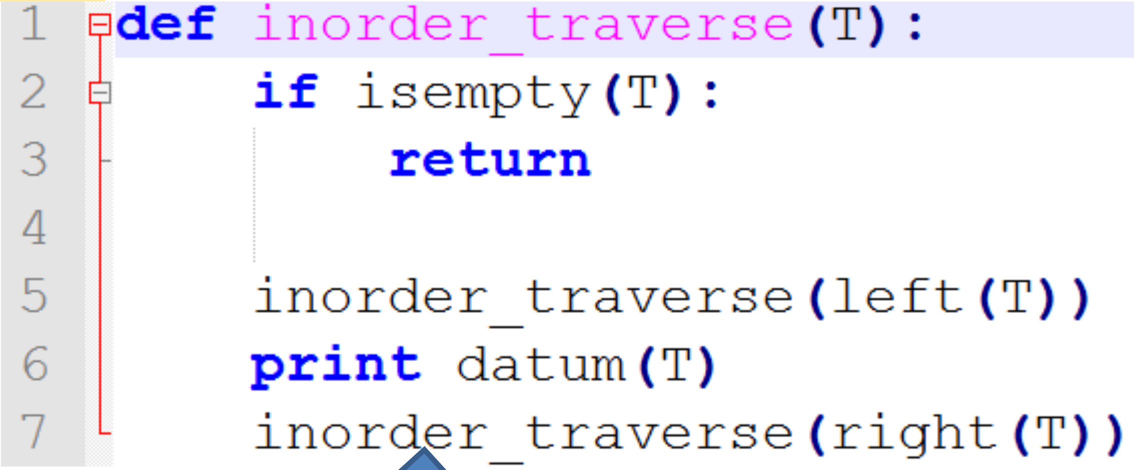
```
def preorder_traverse(T):
    if isempty(T):
        return

    print datum(T)
    preorder_traverse(left(T))
    preorder_traverse(right(T))
```

```
10 5 3 8 30
```

Traversing Trees

2. In-order Traversal



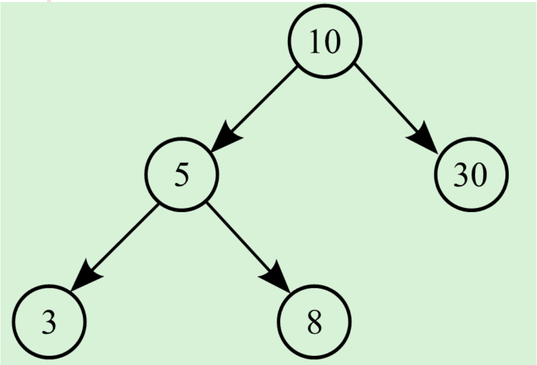


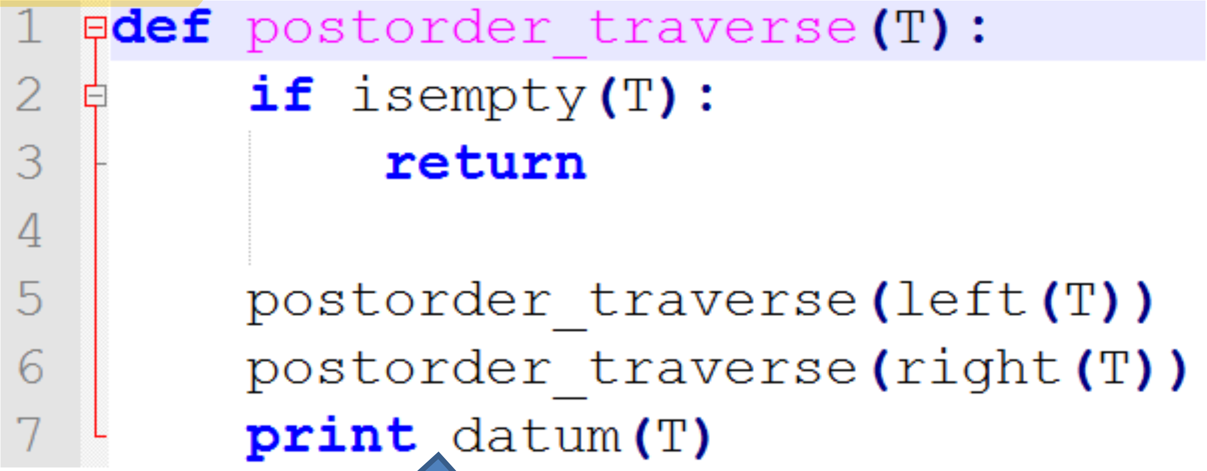
```
def inorder_traverse(T):
    if isempty(T):
        return

    inorder_traverse(left(T))
    print datum(T)
    inorder_traverse(right(T))
```

```
3 5 8 10 30
```

Traversing Trees

3. Post-order Traversal

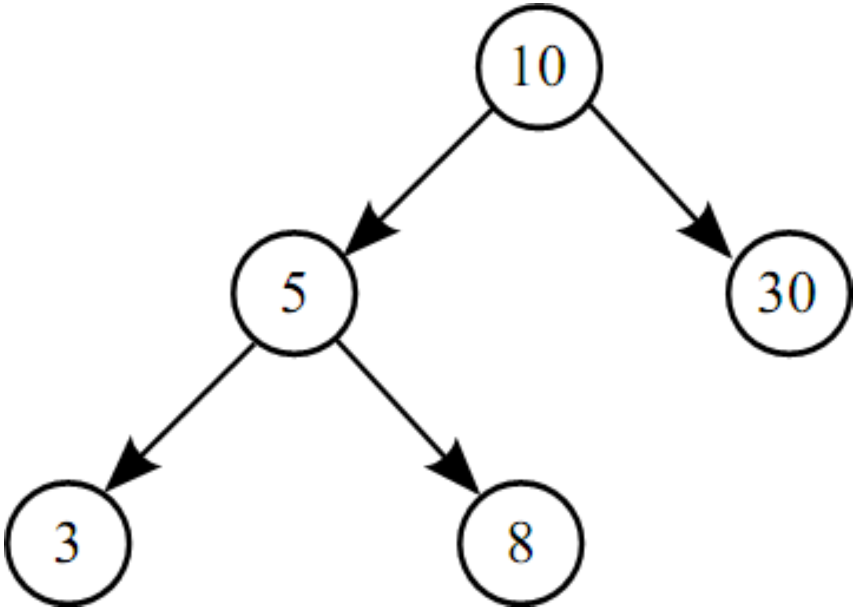




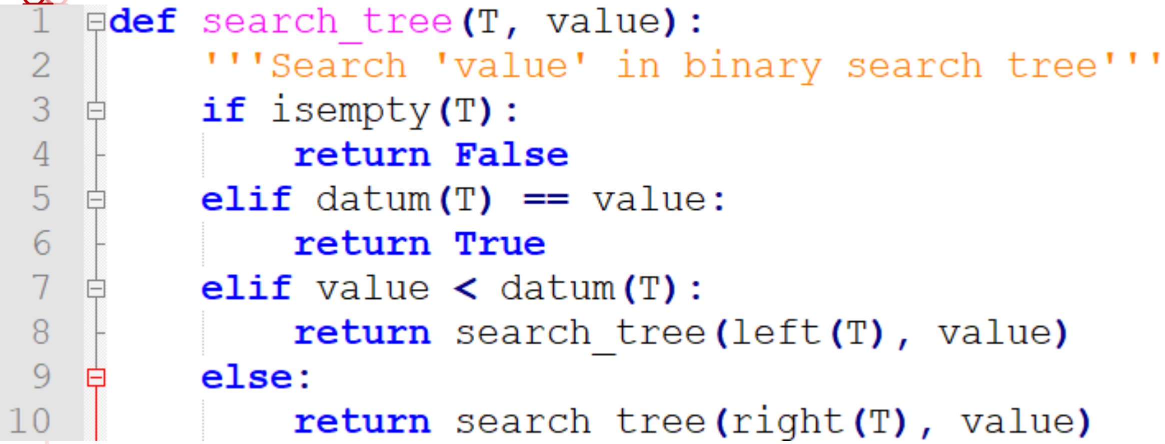
```
def postorder_traverse(T):
    if isempty(T):
        return

    postorder_traverse(left(T))
    postorder_traverse(right(T))
    print datum(T)
```

```
3 8 5 30 10
```

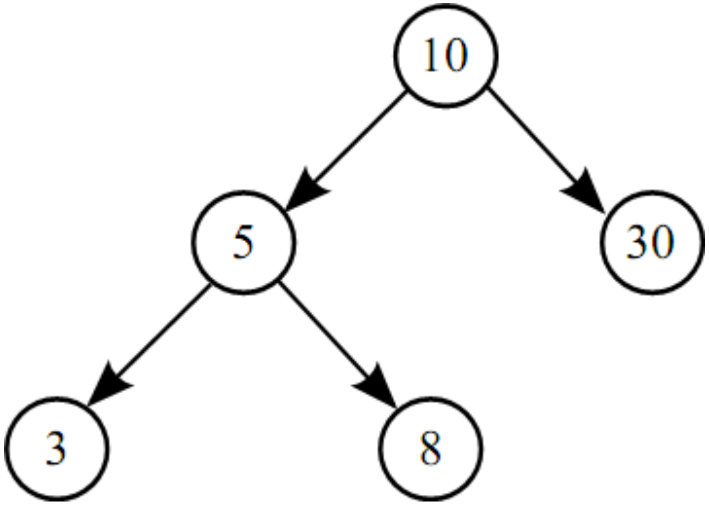
Binary **Search** Trees



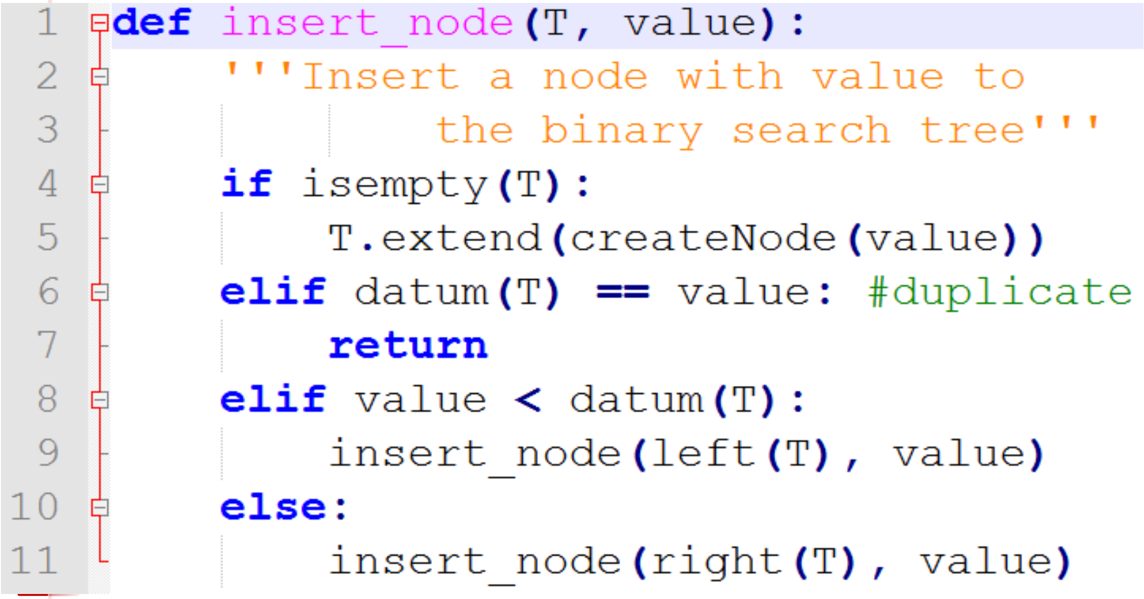
Binary Search Trees: An example



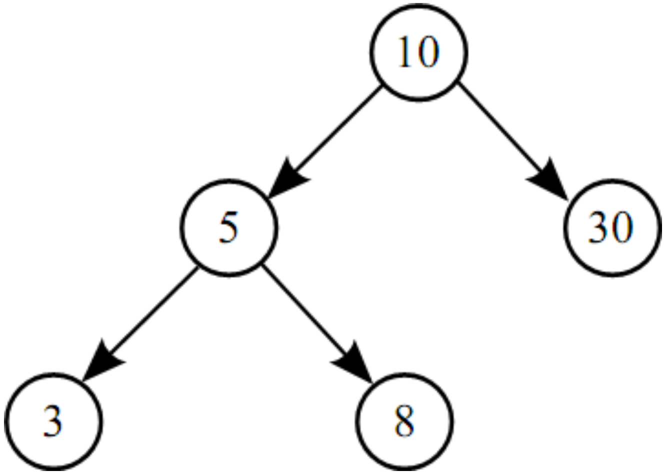
```
def search_tree(T, value):
    '''Search 'value' in binary search tree'''
    if isempty(T):
        return False
    elif datum(T) == value:
        return True
    elif value < datum(T):
        return search_tree(left(T), value)
    else:
        return search tree(right(T), value)
```

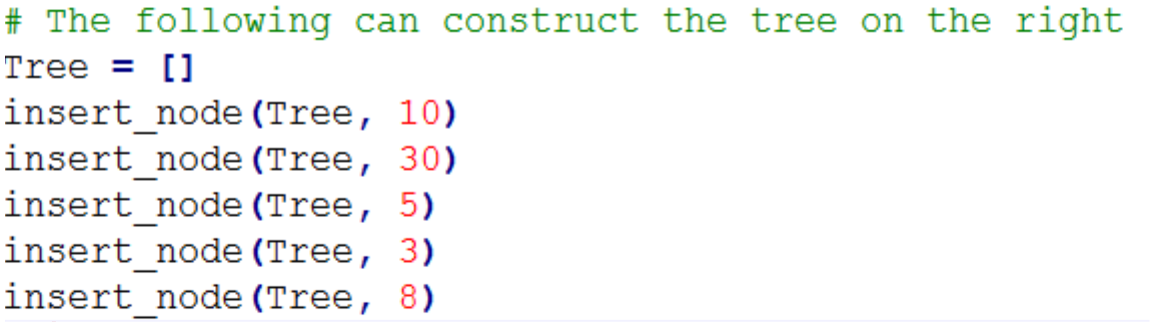


Binary Search Trees: An example



```python
def insert_node(T, value):
    '''Insert a node with value to
            the binary search tree'''
    if isempty(T):
        T.extend(createNode(value))
    elif datum(T) == value: #duplicate
        return
    elif value < datum(T):
        insert_node(left(T), value)
    else:
        insert_node(right(T), value)
```





```python
# The following can construct the tree on the right
Tree = []
insert_node(Tree, 10)
insert_node(Tree, 30)
insert_node(Tree, 5)
insert_node(Tree, 3)
insert_node(Tree, 8)
```

Exercises

1. Write a function to determine the height of a binary tree.
2. Write a function to determine whether a binary tree is balanced (a tree T is balanced if |height(left(T))-height(right(T))|<= 1 for every node in T)
3. Write a function to swap left and right branches of every node.
4. Write a function to count the leaves of a binary tree.