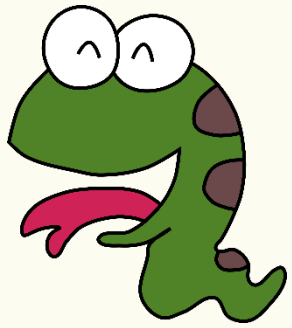
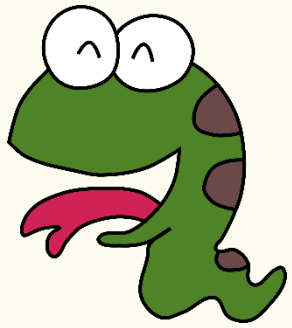


Python Programming

We will be starting shortly

In the meantime, sit back and relax!

(Slides for today's lecture are on Scientia)



Python Programming

Warning!

We will start recording this session now!

Also, any messages in the text chat will remain on MS Teams even after the session

Quiz

Why don’t snakes like to weigh themselves?

Coursework 1

Well done on submitting on time!

Lesson 9 released!

Finally! Sorry for the delay!

Future materials

- Next to be produced: Lesson 10

- Scientific libraries
  - NumPy: See Intro2ML's materials for now
  - pandas/scikitlearn: Less urgently needed for your degree

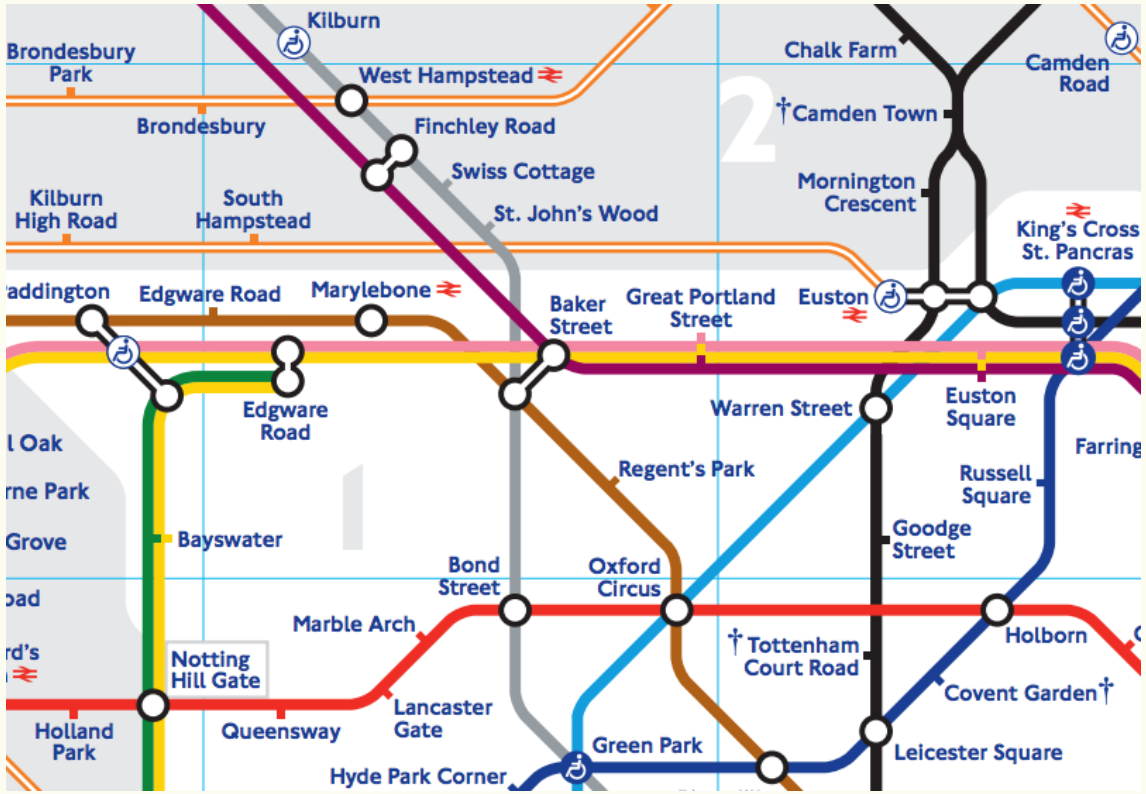
Coursework 2

To be released **tomorrow**

New deadline: **Thurs 28th Oct 7pm BST**

Coursework 2

- 6% of final module grade



Coursework 2

- Tasks (average 5 hours)
  - Complete the **`TubeMap`** class (read JSON file)
  - Complete the **`NeighbourGraphBuilder`** class
  - Complete the **`PathFinder`** class (Dijkstra's algorithm)

Most time

- GitLab Repo/LabTS
  - No more readme.pdf required!
  - Test your own code (no need to show us!)

Coursework 2

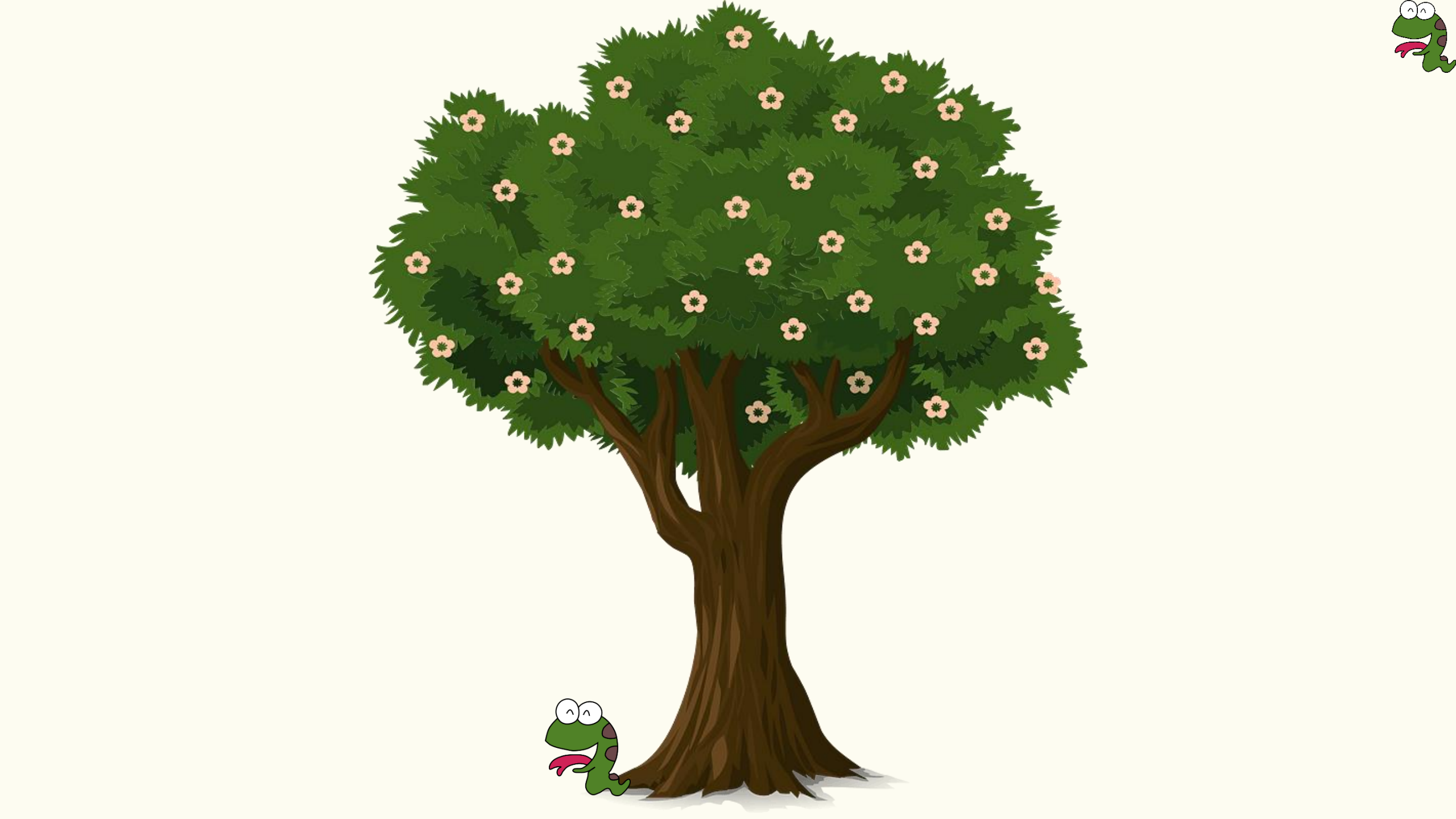
- Lesson 9
  - Directly relevant for CW2
    - Chapter 2 (OOP)
    - Chapter 3 (Refactoring exercise)
    - Chapter 6 (JSON)
    - Chapter 7 (Refactoring exercise – at least read it)
    - Chapter 8 (At least up to 8.3)
  - Others
    - Chapter 4 (enumerate, zip)
    - Chapter 5 (list comprehensions)
    - Chapter 9 (more about exception handling)

Trees

Pre-requisites: Recursion, OOP

Useful: Intro2ML CW1

What is a Tree?



Node

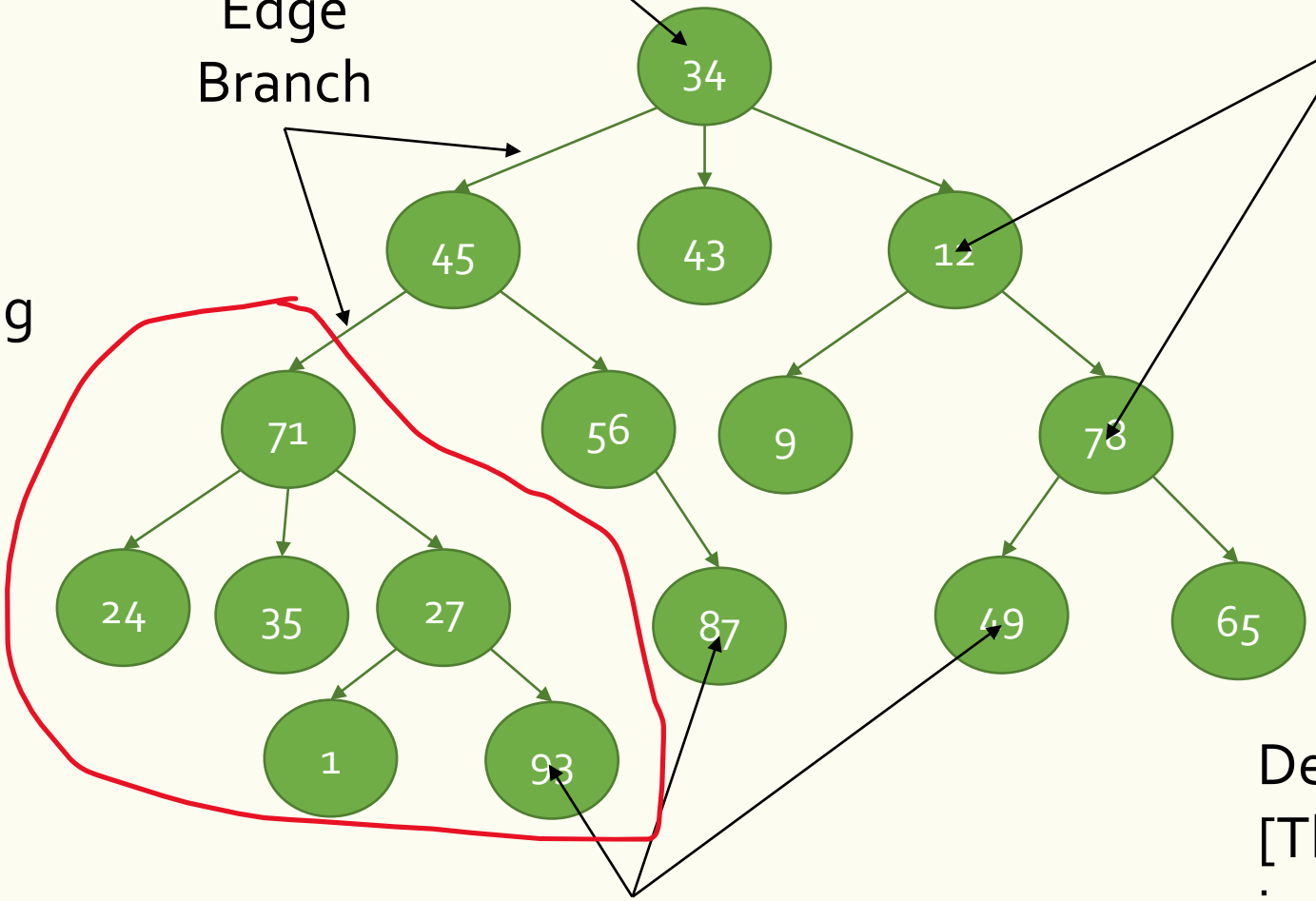
Root node

Edge
Branch

Parent

Child Sibling

Internal node
Non-terminal node
Intermediate node
Branch node



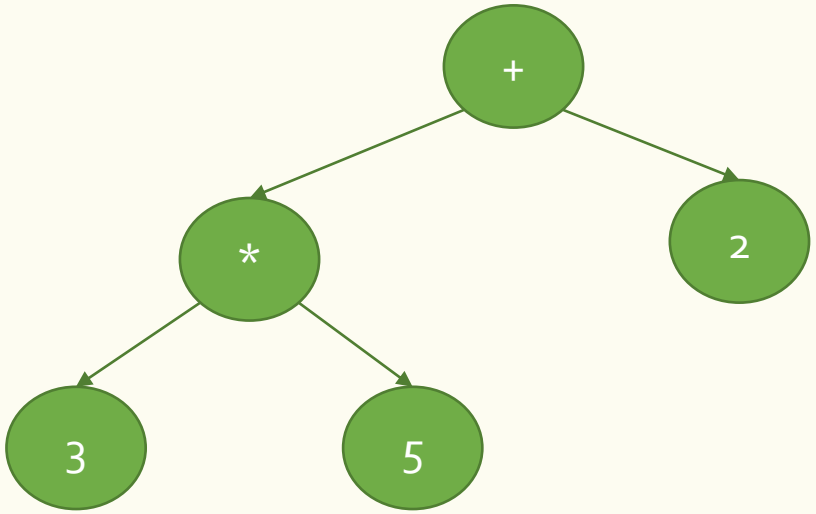
Height
[The maximum the number of edges from root to leaf]

Subtree

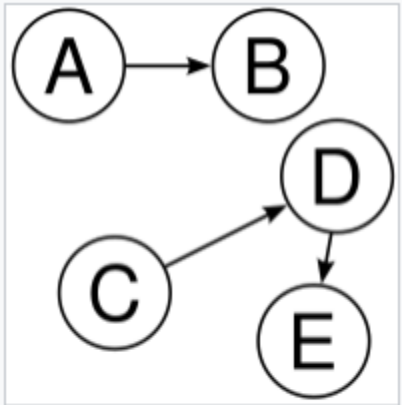
Depth
[The number of edges in the path from a node to the root]

Leaf node
Terminal node

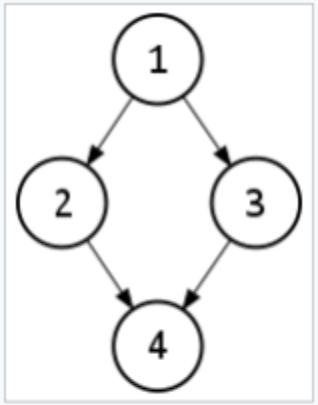
Application: Parsing



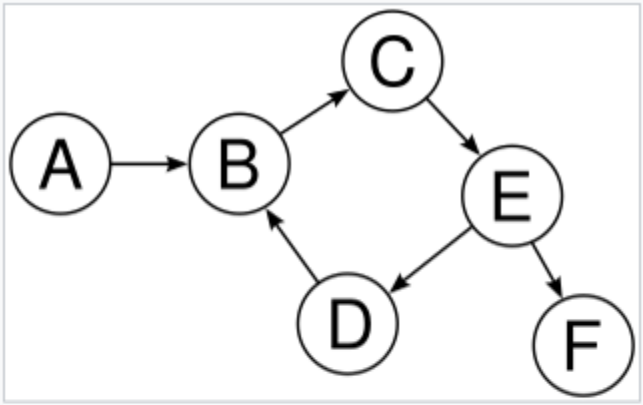
3 * 5 +2



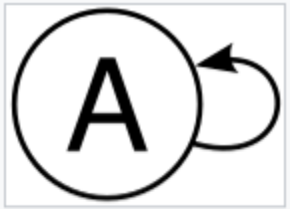
Not a tree: two non-connected parts, A→B and C→D→E. There is more than one root.



Not a tree: undirected cycle 1-2-4-3. 4 has more than one parent (inbound edge).



Not a tree: cycle B→C→E→D→B. B has more than one parent (inbound edge).

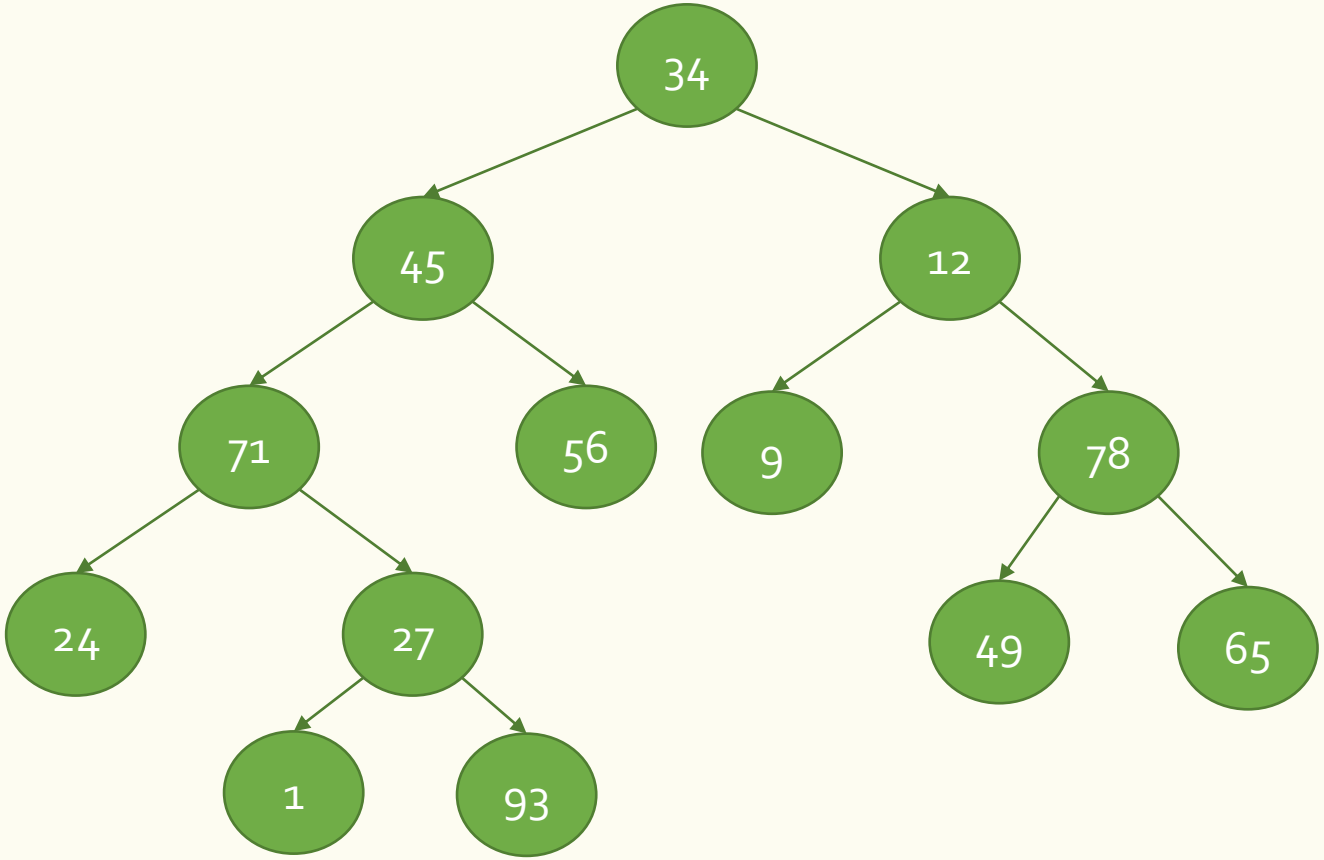


Not a tree: cycle A→A. A is the root but it also has a parent.

Each linear list is trivially a tree

Shamelessly copied from https://en.wikipedia.org/wiki/Tree_(data_structure)

Binary Tree



```
class BinaryTree:
    def __init__(self, value, left_child=None, right_child=None):
        self.value = value
        self.left_child = left_child
        self.right_child = right_child
```

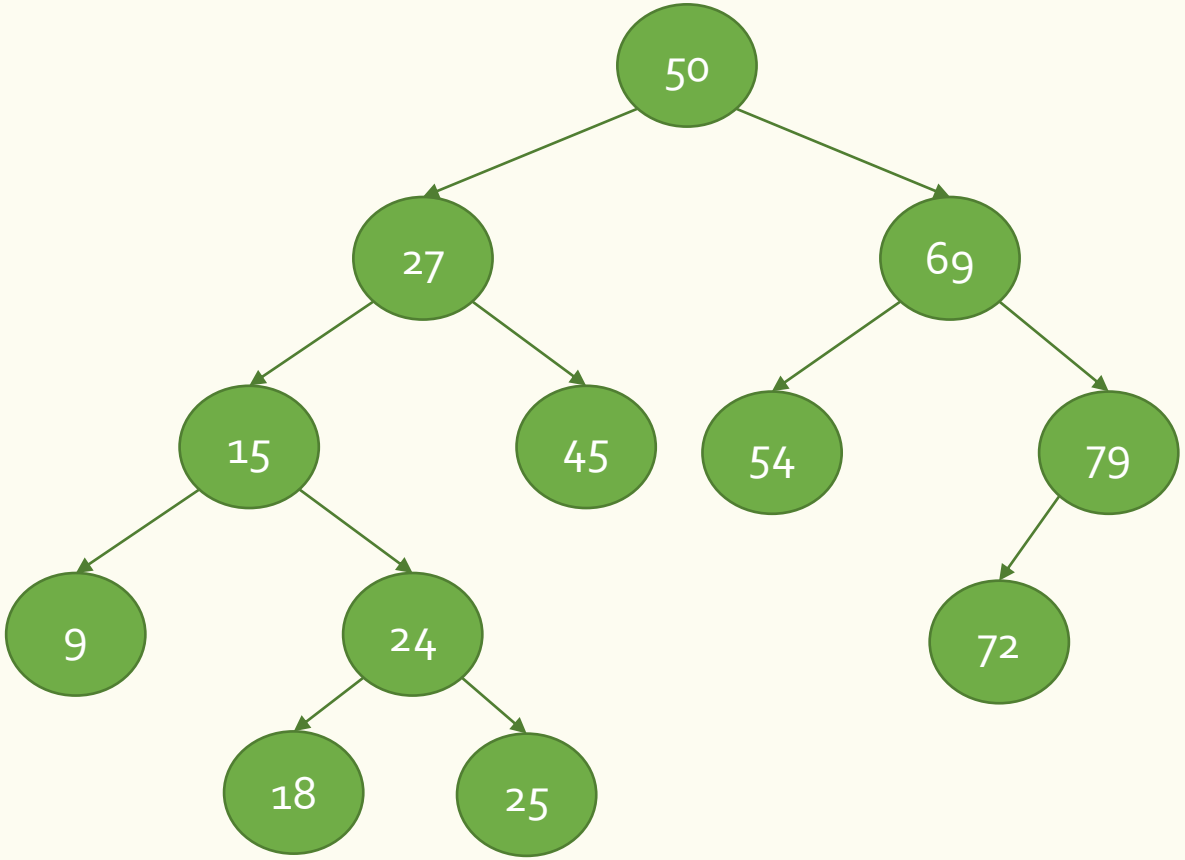
```
class Tree:
    def __init__(self, value, children=[]):
        self.value = value
        self.children = children
```

Binary Search (last week)

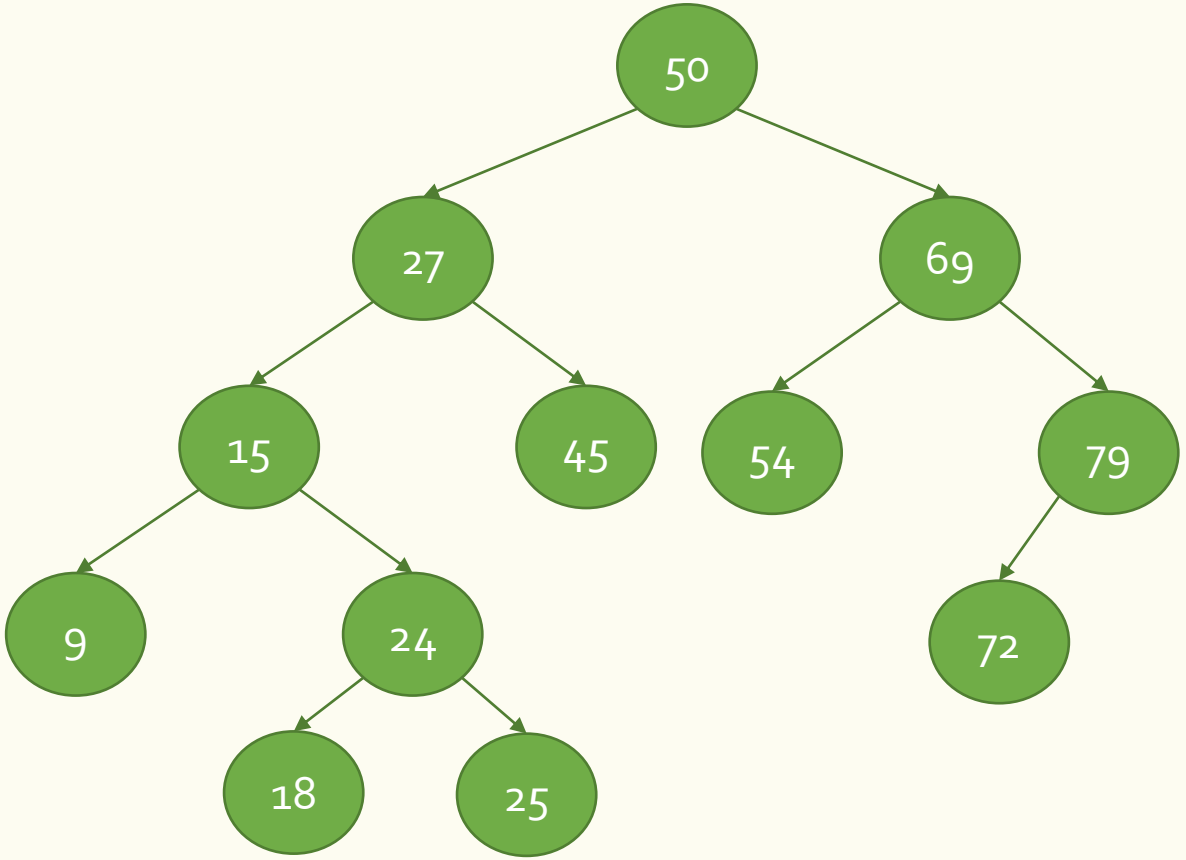
50	69	54	27	15	79	24	18	9	45	72	25

9	15	18	24	25	27	45	50	54	69	72	79

Binary Search Tree



Binary Search Tree: Construction

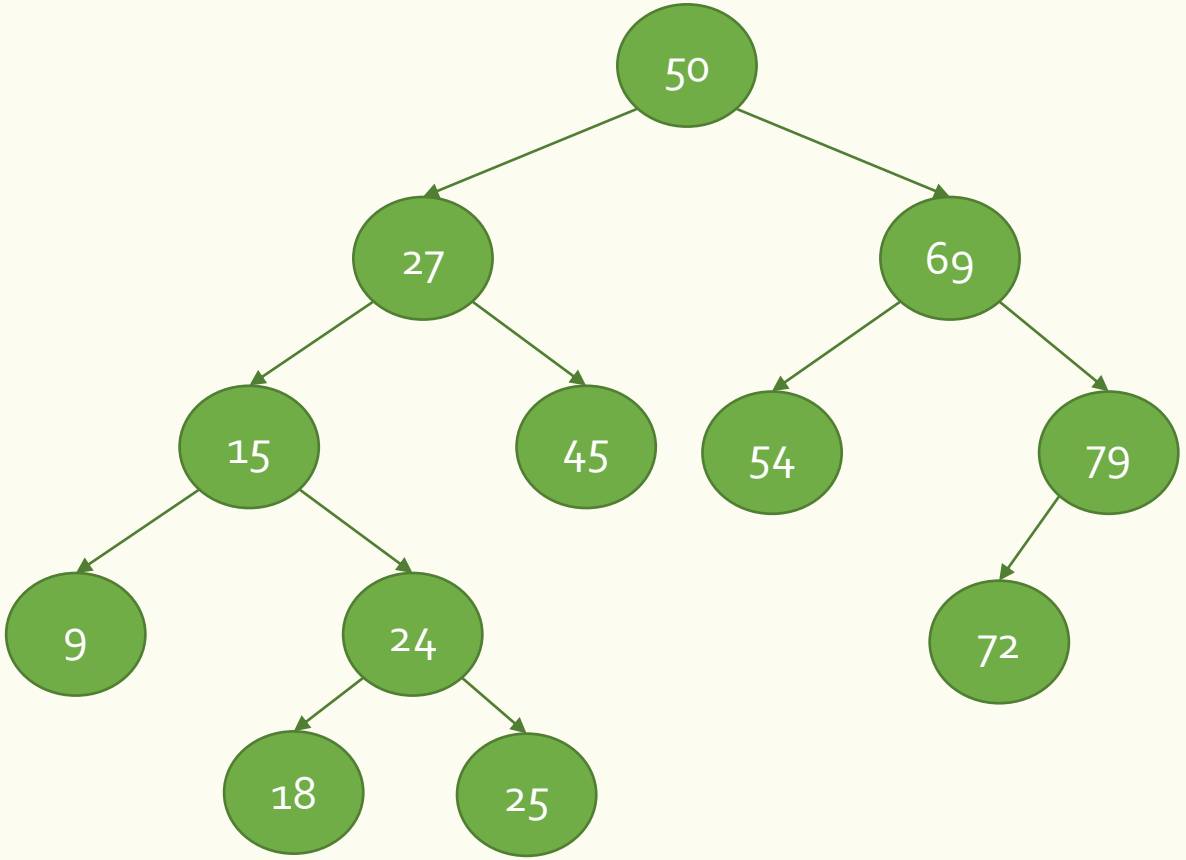


50	69	54	27	15	79	24	18	9	45	72	25

```
class BinarySearchTree:
    def __init__(self, value, left=None, right=None):
        self.value = value
        self.left_child = left
        self.right_child = right
```

```
class BinarySearchTreeBuilder:
    def insert(self, value, node):
        if value <= node.value:
            if node.left_child is not None:
                self.insert(value, node.left_child)
            else:
                node.left_child = BinarySearchTree(value)
        else:
            if node.right_child is not None:
                self.insert(value, node.right_child)
            else:
                node.right_child = BinarySearchTree(value)
```

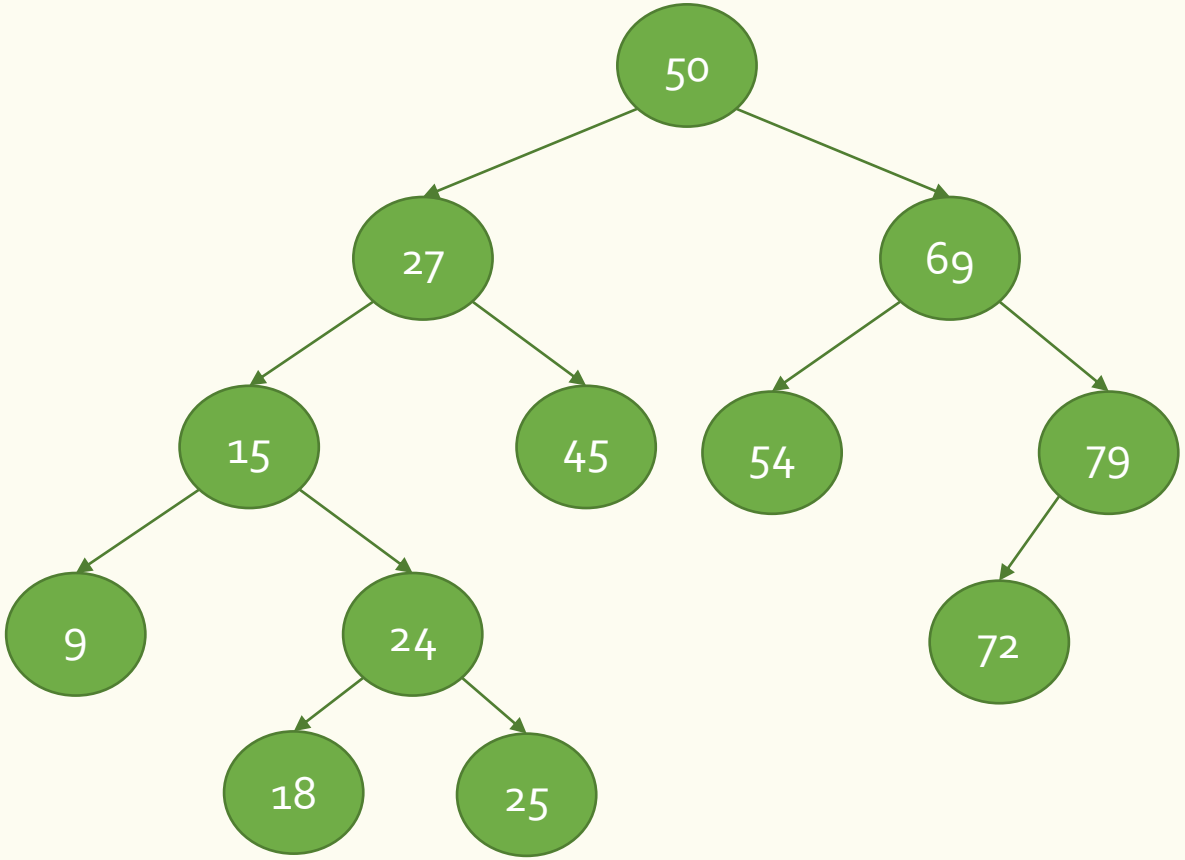
Binary Search Tree: Query



```
class BinarySearchTree:
    def __init__(self, value, left=None, right=None):
        self.value = value
        self.left_child = left
        self.right_child = right
```

```
class BinarySearchTreeFinder:
    def find(self, query, node):
        if query == node.value:
            return True
        elif query <= node.value:
            if node.left_child is not None:
                return self.find(query, node.left_child)
            else:
                return False
        else:
            if node.right_child is not None:
                return self.find(query, node.right_child)
            else:
                return False
```

Binary Search Tree: Traversal



```
class BinarySearchTree:
    def __init__(self, value, left=None, right=None):
        self.value = value
        self.left_child = left
        self.right_child = right
```

```
class BinarySearchTreeWalker:
    def preorder(self, node):
        if node.left_child is not None:
            self.preorder(node.left_child)

        print(node.value)

        if node.right_child is not None:
            self.preorder(node.right_child)
```

Summary

- Trees
  - Definitions
  - Binary trees
  - Binary search trees
    - Construction (Insertion)
    - Query
    - Traversal

This week's schedule

Mon 3-4pm	Mon 4-5pm	Tue 9-10am	Wed 9-10am	Thu 11am-1pm
LECTURE Online only	LAB Online only	LAB 219 + Online	LAB 219 + Online	LAB 221/225 + Online

Next week's lecture topic: OOA&D

One on one with Josiah

- This week on Wed 9am-10am and Thur 11am-1pm
- By appointment
  - https://app.harmonizely.com/josiah-wang/python

Any feedback for us?

- https://www.menti.com/7qxudnnc3i
- Or go to www.menti.com and enter **1011 6313**