Trees

- A non-linear hierarchical data structure
- Can be very efficient
- Example: You look for a file containing your A1 project source code.

  Option 1: you have 10000 files, all stored in a single folder

  Option 2: you have 10000 files organised hierarchical with subfolders,

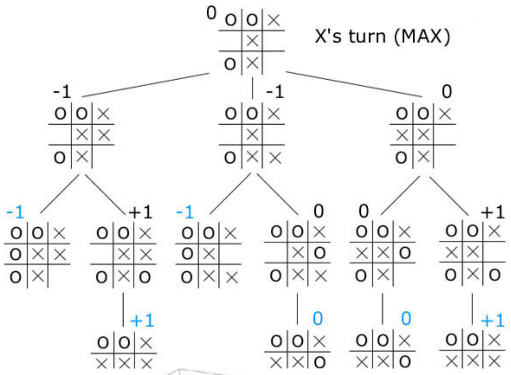
  e.g. university->year1->CS130->A1->SourceCode

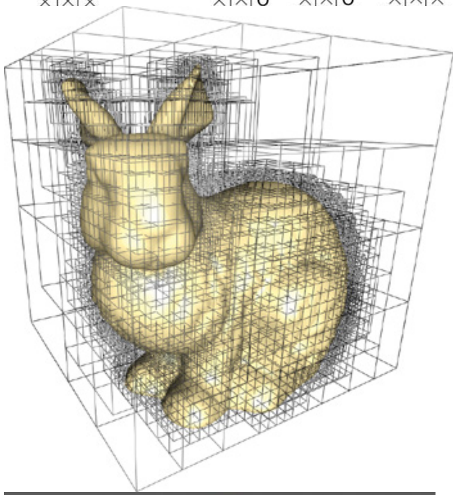
  Which option makes it easier to find the file?

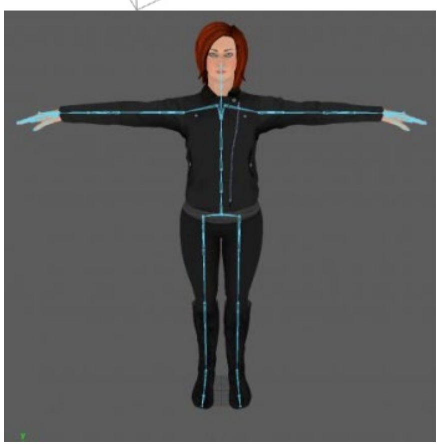
Tree Applications

Examples and Applications of trees include:

- Family trees
- Directory structures (file system)
- Arithmetic expressions
- Game trees (finding winning positions in a game)
- Binary Space Partitioning (BSP) trees (finding visible objects in a scene)
- Search trees (finding all the paths back out of a maze)
- Quadtrees (for efficient terrain rendering)
- Octree (for fast collision detection, ray tracing etc.)
- Constructed Solid Geometry (CSG) objects
- Joint hierarchies (for skeletal animation of characters)
- etc.

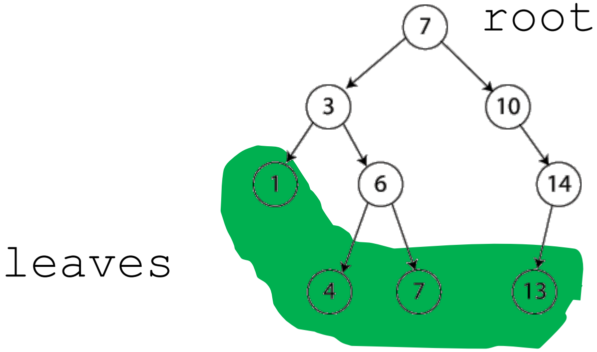






Tree Definition

A tree is defined as a set of points called *nodes* and a set of lines called *edges* where an edge connects two distinct nodes.



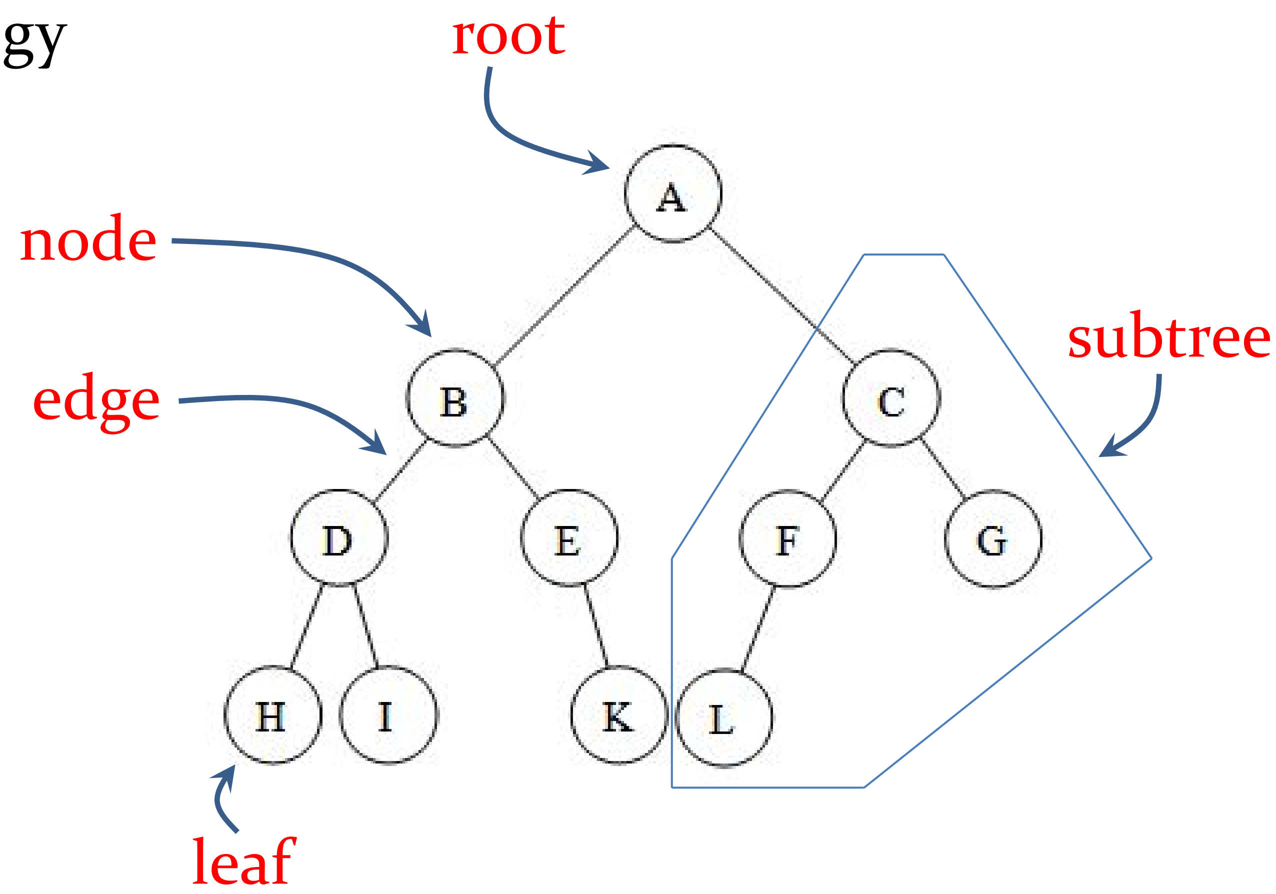
A tree has three properties:

- One node is distinguished and called the root.
- Every node *n* other than the root is connected by an edge to exactly one other node *p* closer to the root.
- A tree is connected in the sense that if we start at any node *n* other than the root and move to the parent of *n*, continue to the parent of the parent of *n*, and so on, we eventually reach the root.

Recursive definition: A tree is either empty or it consist of a node (called root) and a list of subtrees connected to it

Trees

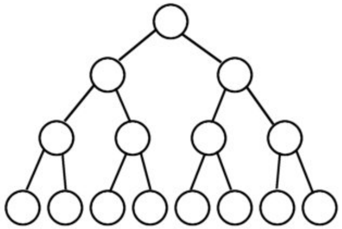
- Some terminology



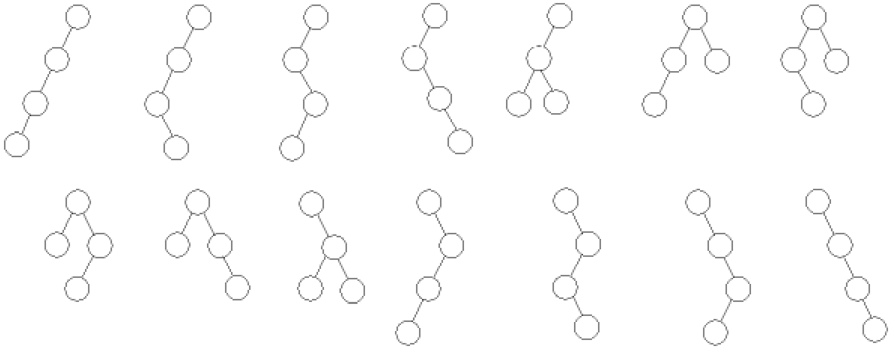
- A node N is the child of M (and M the parent of N) if they are connected by an edge and M is closer to the root than N.
- A tree where every node has at most two children is called a binary tree

Binary Trees

- We normally think of binary trees being balanced and full



- But they can be all sorts of different shapes and sizes

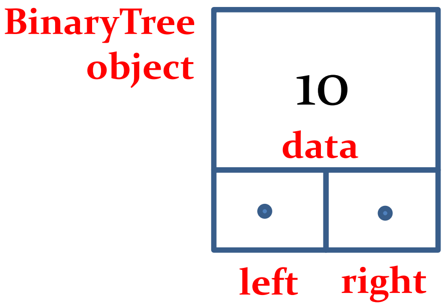


- What tree has only leaf nodes?
  - A tree consisting only of the root, i.e. with no children
- What is the maximum height (number of levels) of a binary tree with 7 nodes? What is its minimum height?
  - Maximum height is 7 (each node, except the last, has one child)
  - Minimum height is 3 (root and children of root have all two children)

BinaryTree class

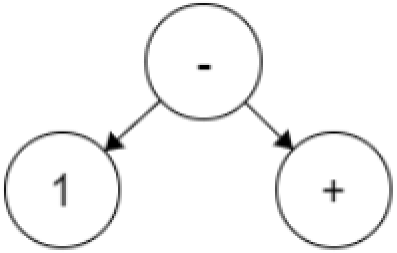
```
class BinaryTree:

    def __init__(self, data, left=None, right=None):
        self.__data = data
        self.__left = left
        self.__right = right
    def insert_left(self, new_data):
        if self.__left == None:
            self.left = BinaryTree(new_data)
        else:
            t = BinaryTree(new_data, left=self.__left)
            self.__left = t
    def insert_right(self, new_data):
        if self.__right == None:
            self.__right = BinaryTree(new_data)
        else:
            t = BinaryTree(new_data, right=self.__right)
            self.__right = t
```

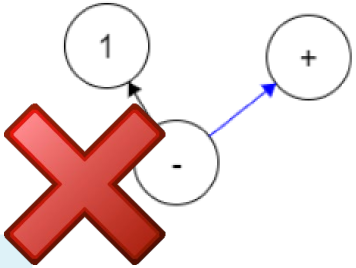
```
def get_left(self):
    return self.__left
def get_right(self):
    return self.__right
def set_data(self, data):
    self.__data = data
def get_data(self):
    return self.__data
...
```

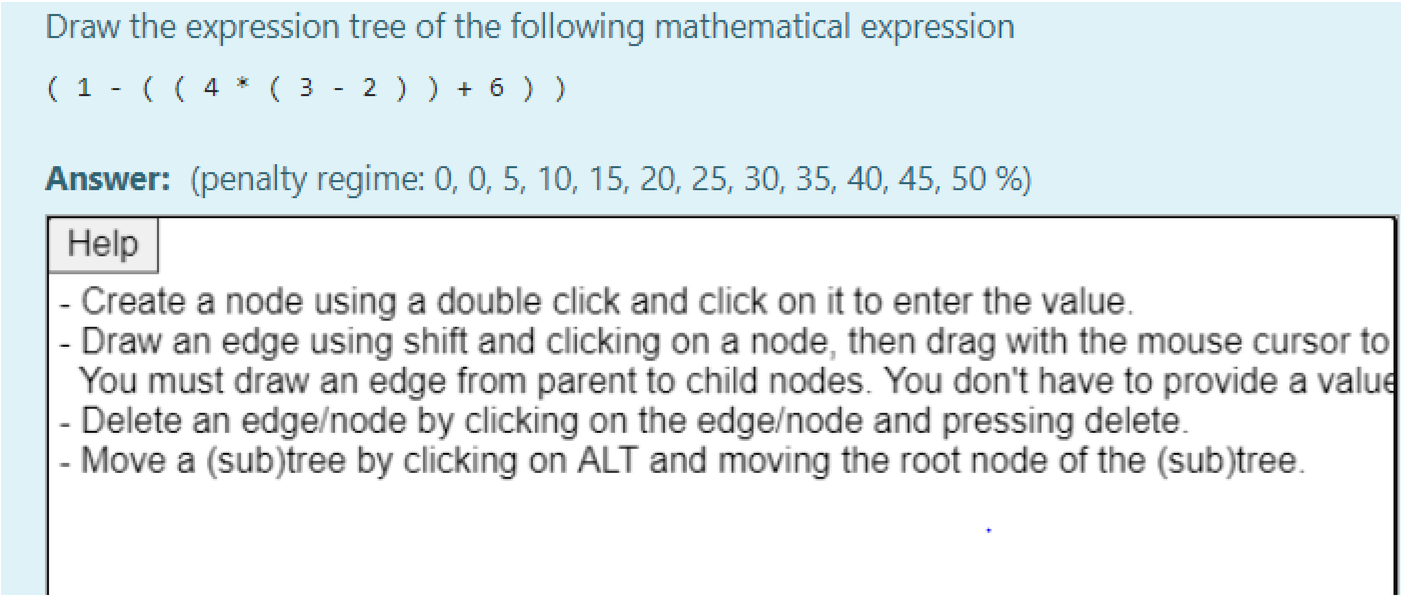


Drawing a tree

- Note:
  - root must be at the top!
  - draw an edge from parent to child nodes






Draw the expression tree of the following mathematical expression

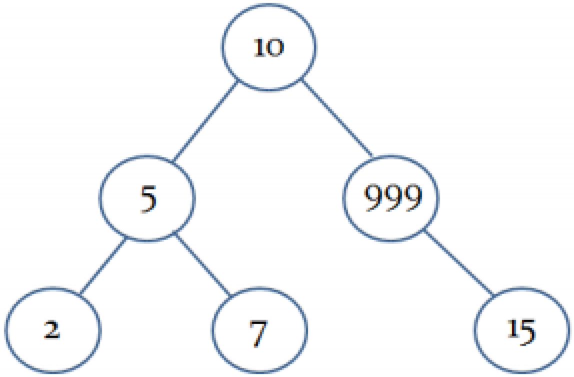
```
( 1 - ( ( 4 * ( 3 - 2 ) ) + 6 ) )
```

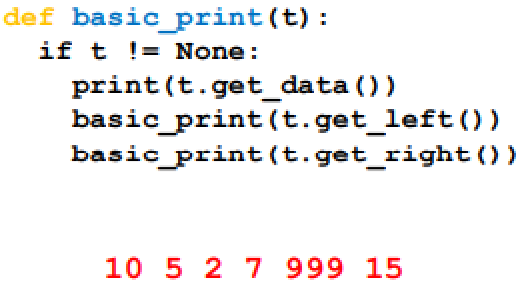
Answer: (penalty regime: 0, 0, 5, 10, 15, 20, 25, 30, 35, 40, 45, 50 %)

- Create a node using a double click and click on it to enter the value.
- Draw an edge using shift and clicking on a node, then drag with the mouse cursor to
  You must draw an edge from parent to child nodes. You don't have to provide a value
- Delete an edge/node by clicking on the edge/node and pressing delete.
- Move a (sub)tree by clicking on ALT and moving the root node of the (sub)tree.

Tree traversals

- The order in which we print the nodes depends on where we place the print() statement relative to the recursive calls

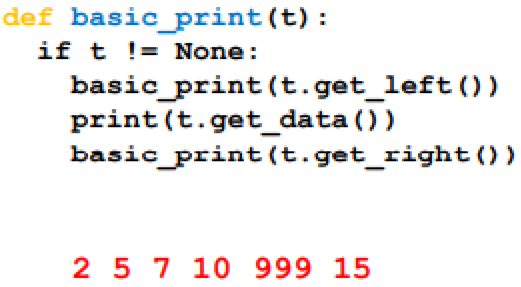




```
def basic_print(t):
  if t != None:
    print(t.get_data())
    basic_print(t.get_left())
    basic_print(t.get_right())
```

```
10 5 2 7 999 15
```

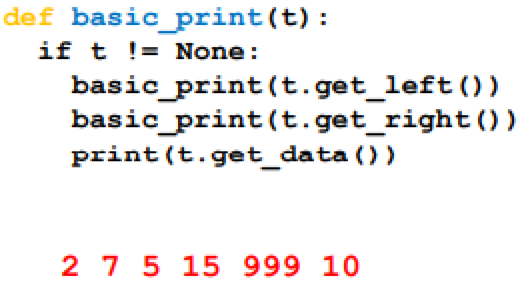
pre-order



```
def basic_print(t):
  if t != None:
    basic_print(t.get_left())
    print(t.get_data())
    basic_print(t.get_right())
```

```
2 5 7 10 999 15
```

in-order



```
def basic_print(t):
  if t != None:
    basic_print(t.get_left())
    basic_print(t.get_right())
    print(t.get_data())
```

```
2 7 5 15 999 10
```

post-order