Lecture 13: Trees

Using a Circle class

```
>>> import circle
>>> c = circle.Circle(1.0)
>>> c
Circle(1.0)
>>> c.radius
1.0
>>> c.diameter()
2.0
>>> c.area()
3.141592653589793
>>>
```

A Circle class

```
1  import math
2
3  class Circle:
4
5      def __init__(self, radius):
6          self.radius = radius
7
8      def diameter(self):
9          return self.radius*2
10
11     def area(self):
12         return math.pi * self.radius**2
13
14     def __repr__(self):
15         return "Circle({})".format(self.radius)
```

A BinaryTree class

```
1   class BinaryTree:
2
3       def __init__(self, root, left=None, right=None):
4           self.root = root
5           self.left = left
6           self.right = right
7
8       def has_left(self):
9           return self.left is not None
10
11      def has_right(self):
12          return self.right is not None
13
14      def leaf(self):
15          """ returns True if and only if this tree is a leaf """
16          return not (self.has_left() or self.has_right())
```

A recursive `all_paths` method

```
1        if self.leaf():
2            return [[self.root]]
3         else:
4            paths = []
5
6            if self.has_left():
7                left_paths = self.left.all_paths()
8                for path in left_paths:
9                    path.insert(0, self.root)
10               paths.extend(left_paths)
11
12           if self.has_right():
13               right_paths = self.right.all_paths()
14               for path in right_paths:
15                   path.insert(0, self.root)
16               paths.extend(right_paths)
17
18           return paths
```

Building a Coin Flip Tree

```
1  from tree import BinaryTree
2
3  def build_tree(n):
4      """
5      Create a binary tree corresponding to all possible
6      outcomes of 'n' tosses of a two-sided coin
7      """
8      if n == 0:
9          return BinaryTree('*')
10     else:
11         left = build_tree(n-1)
12         right = build_tree(n-1)
13         left.root = 'H'
14         right.root = 'T'
15         return BinaryTree('*', left, right)
16
17 def generate_tosses(n):
18     return [path[1:] for path in build_tree(n).all_paths()]
19
20 if __name__ == '__main__':
21     for path in generate_tosses(int(sys.argv[1])):
22     print(" -".join(path))
```