```python
import os

def bigfiles(dirname, min_size):
    large = []
    #print dirname
    for sub in os.listdir(dirname):
        path = os.path.join(dirname, sub)
        if os.path.isdir(path):
            large.extend(bigfiles(path, min_size))
        else:
            size = os.path.getsize(path)
            if size > min_size:
                large.append((path, size))
    return large

def ftree(indent, dirname, depth):
    sub_depth = indent.count("−")
    if sub_depth >= depth:
        return
    print indent, dirname
    contents = os.listdir(dirname)
    for sub in contents:
        path = os.path.join(dirname, sub)
        if os.path.isdir(path):
            ftree(indent + "−−−", path, depth)
        else:
            print indent, path

if __name__ == "__main__":
    ftree("−", "/Users/ola/Desktop", 3)
    bigs = bigfiles("/Users/ola/Desktop/courses/6python", 10000)
    for f in bigs:
        print f
```

```python
import turtle

def draw(iters):
    flake = "FRFRF"
    for steps in range(iters):
        flake = flake.replace("F", "FLFRFLF")
        for move in flake:
            if move == "F":
                turtle.forward(100.0/3**(iters-1))
            elif move == "L":
                turtle.left(60)
            elif move == "R":
                turtle.right(120)
    draw(2)
    x = raw_input()
```
```python
import random

def merge(left, right):
    result = []
    i, j = 0, 0
    while i < len(left) and j < len(right):
        if left[i] <= right[j]:
            result.append(left[i])
            i += 1
        else:
            result.append(right[j])
            j += 1
    result.extend(left[i:])
    result.extend(right[j:])
    return result

def mergesort(list):
    if len(list) < 2:
        return list
    else:
        middle = len(list) / 2
        left = mergesort(list[:middle])
        right = mergesort(list[middle:])
        return merge(left, right)

def isSorted(lst):
    for i, n in enumerate(lst[1:]):
        if n < lst[i]:
            return False
    return True

nums = [x for x in range(0,100)]
random.shuffle(nums)
snums = mergesort(nums)

print isSorted(nums)
print isSorted(snums)
```

```python
import Image
import ImageDraw
import random

def midpoint(p,q):
    return ((p[0]+q[0])/2.0,(p[1]+q[1])/2.0)

def sierpinski(p1,p2,p3, image, level):
    '''calculates points for sub triangles, uses recursion for steps'''
    if level <= 0:
        return
    # draw triangles each step through
    image.line([p1,p2])
    image.line([p2,p3])
    image.line([p1,p3])

    # make smaller triangles from each point and adjacent midpoints
    sierpinski(p1,midpoint(p1,p2),midpoint(p1,p3),image,level−1)
    sierpinski(p2,midpoint(p2,p3),midpoint(p2,p1),image,level−1)
    sierpinski(p3,midpoint(p3,p2),midpoint(p3,p1),image,level−1)

def chaos(p1,p2,p3,image,iters):
    plist = [p1,p2,p3]
    current = (random.uniform(p1[0],p2[0]),.2)
    for x in xrange(iters):
        mid = midpoint(current,vert)
        image.point(mid,fill='black')
        current = mid

# starting point for equilateral triangle
tpoints = ((0, 700), (700, 700), (350, 0))
level = 8
size = tpoints[1]
picture = Image.new('RGB', size,color='blue')
picture2 = Image.new('RGB', size,color='white')
drawimage = ImageDraw.Draw(picture)
drawimage2 = ImageDraw.Draw(picture2)

for x in tpoints:
    vert = random.choice(plist)

# make smaller triangles from each point and adjacent midpoints
sierpinski(p1,midpoint(p1,p2),midpoint(p1,p3),image,level−1)
```
import random

def create_content():
    """
    return a dictionary used for generating random sentences
    """
    adjectives = ["<color>*, "slimy", "wonderful", "beautiful", "obese", "teeny", "<adj> . <adj>"]
    colors = ["green", "red", "yellow", "blue", "maroon"]
    rules = ["<color>:colors, "<adj> :adjectives"

    return rules

def expand(sentence, rules):
    """
    expand sentence using rules as source of meta-words
    """
    sent = ""
    for w in sentence.split():
        if w.startswith("<"):  
            chosen = random.choice(rules[w])
            sent += expand(chosen, rules) +"
        else:
            sent += w + ""
    return sent.strip()

def create():
    rules = create_content()
    print expand("the <adj> dog ate the <adj> bone", rules)
    print expand("the <color> house was a <adj> edifice", rules)

if __name__ == "__main__":
    create()