```
# Task 1 Install numpy/scipy/matplotlib
# Task 2 Install the library PIL
# Task 3 Install the library openCV
import scipy
import matplotlib
from PIL import Image, ImageDraw, ImageFont
import numpy as np
import cv2
import sys
#set global color green for the trees
GREEN = "\033[92m"

# Task 4.1 Return to the future
# open the mario.png and convert to a np array
mario_image = Image.open('mario.png')
iar = np.asarray(mario_image)


# Task 4.2 Santa Claus Time
#Create the class Tree
class Tree:
    def __init__(self, leaves, trunk, root):
        self.leaves = leaves
        self.trunk = trunk
        self.root = root
# getters and setters for all of the Tree arguments
    def get_leaves(self):
        return self.leaves

    def set_leaves(self, leaves):
        self.leaves = leaves

    def get_trunk(self):
        return self.trunk

    def set_trunk(self, trunk):
        self.trunk = trunk

    def get_root(self):
        return self.root

    def set_root(self, root):
        self.root = root

    def __repr__(self):
        return "Tree:{} leaves:{} trunk:{} root:{}".format('tree', self.leaves, self.trunk, self.root)

# sub class ChristmasTree
class ChristmasTree(Tree):
    def __init__(self, height, leaves, trunk, root):
        self.height = height
```

```
        # super(Tree, self).__init__(leaves, trunk, root)
        Tree.__init__(self, leaves, trunk, root)
# getter and setter for height
    def get_height(self):
        return self.height

    def set_height(self, height):
        self.height = height

    def __repr__(self):
        return "Christmas tree-> leaves:{}, Height:{}".format(self, self.leaves, self.height)
    #DisplayTree function
    # from http://stackoverflow.com/questions/26555986/using-for-loops-to-create-a-christmas-tree
    def DisplayTree(self):
        z = self.height - 1
        x = 1
        line = ''
        for i in range(self.height):
            line = line + (' ' * z + self.leaves * x + ' ' * z + '\n')
            x += 2
            z -= 1

        print GREEN + line
        return line

# Set the variables for ctree and store it as ctree_image
ctree = ChristmasTree(height=7, leaves='*', trunk=1, root=1)
ctree_image = ctree.DisplayTree()


# Square Tree is also a subclass of Tree
class SquareTree(Tree):
    def __init__(self, height, leaves, trunk, root):
        self.height = height
        Tree.__init__(self, leaves, trunk, root)
# getter/setter and repr string statement
    def get_height(self):
        return self.height

    def set_height(self, height):
        self.height = height

    def __repr__(self):
        return "Square tree-> leaves:{}, Height:{}".format(self, self.leaves, self.height)
# DisplayTree function for a Square tree
    def DisplayTree(self):
        line = ''
        for i in range(self.height):
            line = line + (self.leaves * self.root + '\n')
        print GREEN + line
        return line
```

```
# Assign the variables for stree and store it's output as stree_image
stree = SquareTree(height=5, leaves='$', trunk=1, root=10)
stree_image = stree.DisplayTree()


# Oval tree is also a child class of Tree
class OvalTree(Tree):
    def __init__(self, height, leaves, trunk, root):
        self.height = height
        Tree.__init__(self, leaves, trunk, root)
# getter/setter and repr statement
    def get_height(self):
        return self.height

    def set_height(self, height):
        self.height = height

    def __repr__(self):
        return "Oval tree-> leaves:{}, Height:{}".format(self, self.leaves, self.height)
# DisplayTree function for an Oval Tree
    def DisplayTree(self):
        line = ''
        for i in range(self.height):
            if i == 0:
                line = line + '   ' + (self.leaves * (self.trunk/2)) + '' + '\n'
            else:
                line = line + (self.leaves * self.trunk + '\n')
        line = line + '   ' + (self.leaves * (self.trunk/2)) + '' + '\n'
        print GREEN + line
        return line
#set the attributes for the oval tree and store as otree_image
otree = OvalTree(height=8, leaves='#', trunk=8, root=4)
otree_image = otree.DisplayTree()


# 4.3 mario likes trees
# Create a function that will take the tree images and add them
# to the mario picture using ImageDraw and ImageFont modules
# Save the new image as "mario_looking_at_trees"
def combine_mario_trees(c_tree, s_tree, o_tree):
    draw = ImageDraw.Draw(mario_image)
    font = ImageFont.truetype("arial.ttf", 50)
    draw.text((750, 100), c_tree, (0, 255, 0), font=font)
    draw.text((1150, 300), s_tree, (0, 255, 0), font=font)
    draw.text((1250, 700), o_tree, (0, 255, 0), font=font)
    mario_image.show()
    mario_image.save("mario_looking_at_trees.jpg")

# Call the function combine_mario_trees inputing the 3 tree images
combine_mario_trees(ctree_image, stree_image, otree_image)
```

```
# 4.4 Thresholding

# create a function that creates a list of the means
def list_of_means(iar):
    meanslist = []
    for i in iar:
        for b in i:
            meanslist.append(np.mean(b))
    return meanslist

# create an inversion function
def invert_colors(iar, threshold):
    iar.flags.writeable = True
    for outer_list in iar:
        for pixel in outer_list:
            # print pixel
            if np.mean(pixel) >= threshold:
                # white pixel
                # print pixel
                pixel[0] = 255
                pixel[1] = 255
                pixel[2] = 255
                pixel[3] = 255
            else:
                # black pixel
                # print pixel
                pixel[0] = 0
                pixel[1] = 0
                pixel[2] = 0
                pixel[3] = 255
    return iar


# Define the theshold which is the mean of the list of means
threshold = np.mean(list_of_means(iar))
print(threshold)
# Use the inversion function and turn that array into an image.
# Save as inverted.png
data = invert_colors(iar, threshold)
img = Image.fromarray(data)
img.save('inverted.png')
img.show()
```