uvm_tb_arch_doc_py

Table of contents:

1. Introduction

uvm_tb_arch_doc_py is a python project to automatically generates the UVM testbench Architecture

The main aim of this project is, by using the Python Programming language we have to generate the UVM testbench Architecture template. For generating the UVM TB Architecture we have to write an example testbench code (top, test, env, agent etc) in UVM Methodology.

For generating the UVM Testbench template, by using Python Programming we can use either Python Turtle graphics or Python Image draw graphics

- Python turtle Graphics is used to create shapes and patterns needs to import python turtle from python library, by using Python turtle can generate different shapes but before going to use this Turtle library methods initially we have to define these methods it makes the Turtle graphics as inefficient

- The Image Draw module provides simple 2D graphics support for Image Object. These graphics interface uses the same coordinate system as PIL.PIL is the Python Imaging Library which provides the python interpreter with Image Editing Capabilities. It is simpler and easier to understand it can be install by pip(pip install Pillow) . The pillow color schemes we use is RGB. The color RGB representation and support is provided by the module **ImageColor**.

How to Run python files

Step1: Download the latest python version in Desktop

Step2: Get Pycharm tool or can use command prompt window

Step3: Once the Pycharm tool installed in Desktop, go to file create a project

Step4: PyCharm enables programmers to write high-quality Python code in code editor. The editor enables programmers to read code easily through color schemes. Write an example code in the code editor and then save the file in the project

Step5: Right click on the file then select run option and execute the code

2. Overview of the Modules

Module	Description
dummy_tb	In this folder, simple Memory testbench module code is written in UVM Methodology. Components that are used for building the Memory testbench module are mem_seq_item, mem_wr_seq, mem_rd_seq, mem_seqr, mem_drv, mem_mon, mem_agent, mem_env, mem_test, mem_top.
TB_arch_using_image_draw.py	This TB_arch_using_image_draw script module defines , • How the TB Architecture Blocks (like top, test, env, dut, scoreboard, interface, virtual interface, agent, monitor, driver, seqr) should be generate in the image format with respect to given input co-ordinates && colours • It also defines, how the connections are happened between the Driver to Virtual Interface, Virtual interface to Interface, Interface to DUT and Virtual interface to Driver
component_name_finder.py	This component_name_finder script modules defines, • When the user wants to find particular component in the TB Architecture, first user have to enter the input as the TB Directory path and then enter the Keyword of the particular component name which want to search • Then the Output of this script generates the user selected TB component class name and prints the output in Normal && Table format
draw_rect_using_turtle.py	This draw_rect_using_turtle script module defines • how to draw the rectangle shape by using Turtle library

Module	Description
draw_rect_with_text_inside.py	This draw_rect_with_text_inside scripting module defines • How to write the text inside the rectangle shape by using Image Draw module
pattern_finder_indir.py	This pattern_finder_indir Script defines, • How to find a pattern in all files of given directory i.e., how to find the UVM components from the given test-bench directory • Then it prints the filename and the path of the file. This can be used to find all the different components of a testbench.
skeleton of component search python	This skeleton_component_search script defines • How to find the generic word in TB components that can be searched to determine the presence of component in TB
tb_arch_img_draw.py	This tb_arch_img_draw script defines by using Image, Image draw Module • How to draw the agent components by reading the tb_info.txt file, these file which contains the information of agent i.e., no of agents
uvm_tb_arch_agent.py	This uvm_tb_arch_agent script defines • How to support number of agents upto 7 in the TB Architecture

Module	Description
uvm_tb_name_image_file_generator.py	This uvm_tb_name_image_file_generator script defines • Here user enters the path of TB directory. Setting the width and height according to the screen size and Setting the canvas size, create lookup table for uvm component and name. • Create file for writing component and component name, also clean file before writing the data. ➢ Drawing top level structure, set the co-ordinates and set color for top as yellow color. ➢ Drawing test level structure, set the co-ordinates and set color for top as orange color. ➢ Drawing env level structure, set the co-ordinates and set color for top as yellow color. ➢ Drawing scoreboard level structure, set the co-ordinates and set color for top as red. ➢ Drawing agent level structure, set the co-ordinates and set color for top as blue color also set the image font. ➢ Calling the rectangular creation function. Here, user have to enter the input TB directory path then it will create the block diagram of component TB Architecture like-top, test, env, agent, scoreboard

3. Description of each Module

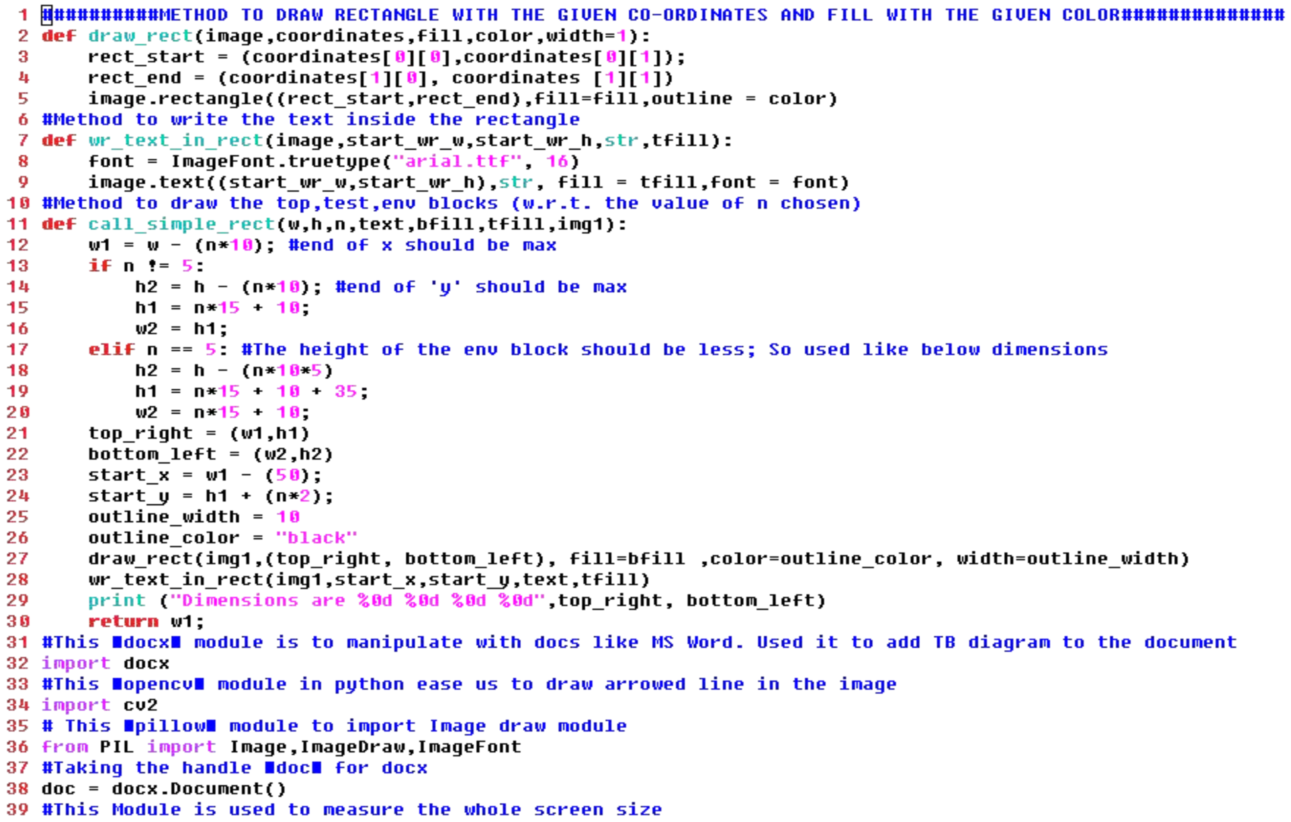
3.1 TB_arch_using_image_draw.py

- To draw TB Diagram aimed at 2 agentsTo draw Rectangle with given co-ordinate & fill with given colour and write the text inside rectangle

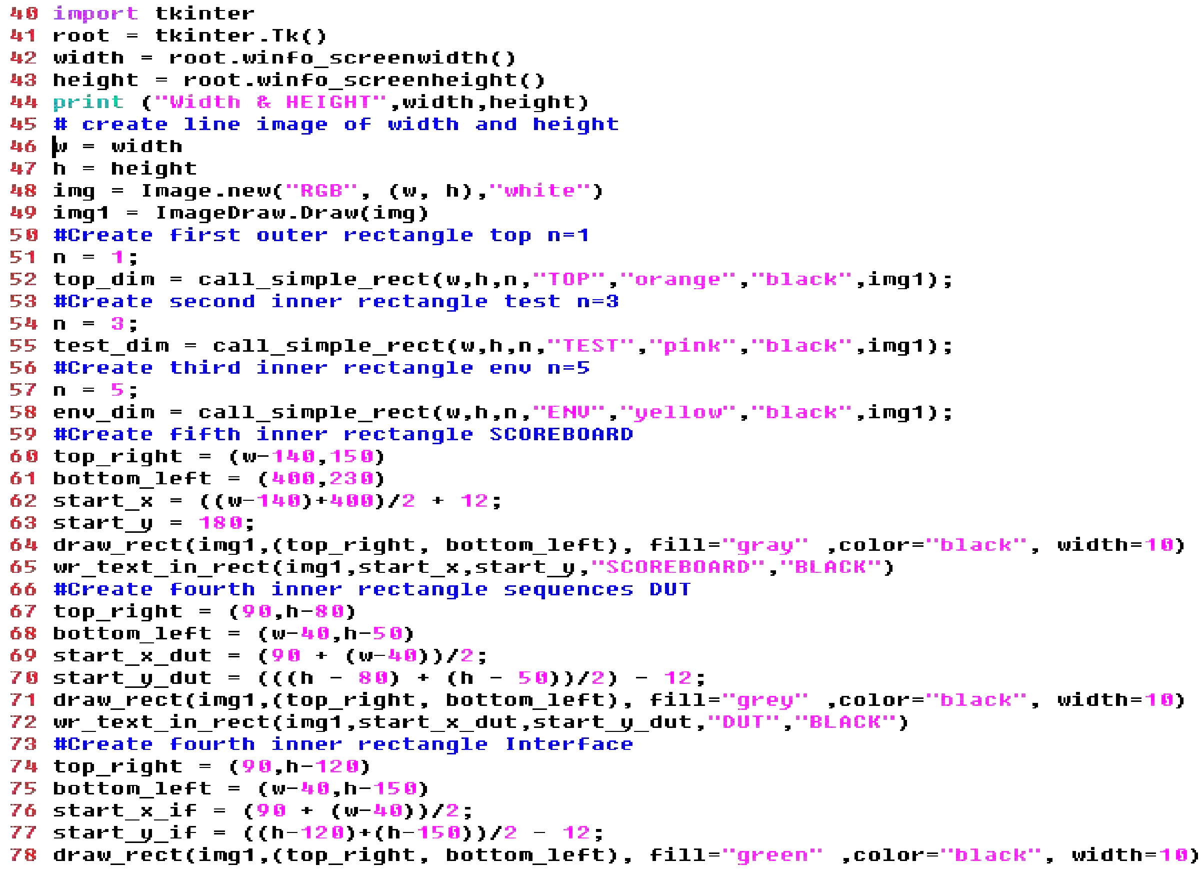
- To draw top, test, env blocks write the value of n choosen height of env block should be less , so we have to give proper dimensions
- This “docx” module is to manipulate with docs like MS Word. Used it to add TB diagram to the document, we have take handle doc for docx

- After we have to import tkinter & setting height & width to measure whole screen size and then
    - Create 1st outer rectangle for top
    - Create 2nd inner rectangle for test
    - Create 3rd inner rectangle for env
    - Create 5th inner rectangle for scoreboard
    - Create 4th inner rectangle for sequences, DUT, Interface, Virtual Interface
    - Check for No of agents user have to give
    - Start another rectangle MON inside agent
    - Start another rectangle DRV inside agent
    - Start another rectangle SEQR inside agent

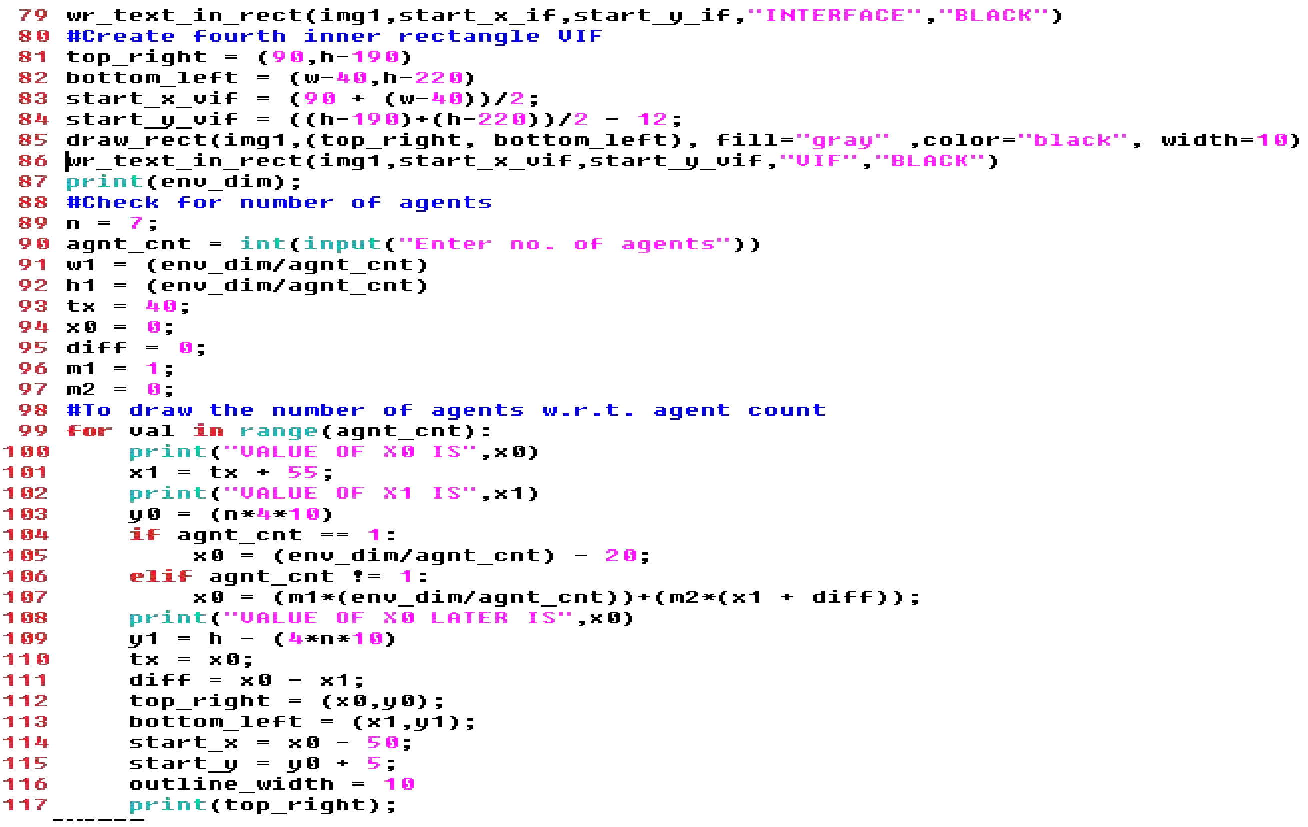
- Then draw the arrows between Driver to Virtual Interface, Virtual Interface to Interface and Driver1 to Virtual Interface and then set the start & end co-ordinates
- By using arrowed line() method set the colour & thickness and then add the picture & save the picture in docx



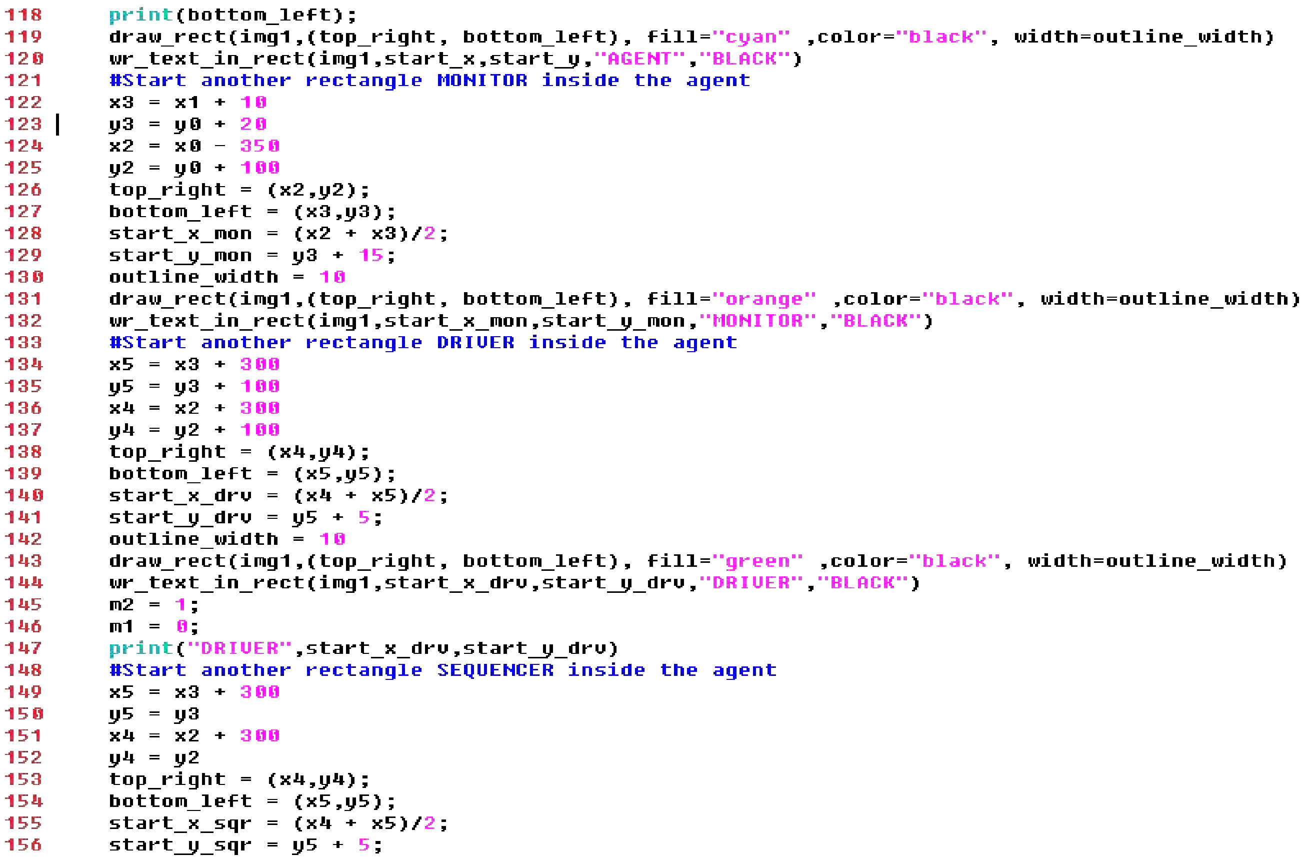
```
##########METHOD TO DRAW RECTANGLE WITH THE GIVEN CO-ORDINATES AND FILL WITH THE GIVEN COLOR##############
def draw_rect(image,coordinates,fill,color,width=1):
    rect_start = (coordinates[0][0],coordinates[0][1]);
    rect_end = (coordinates[1][0], coordinates [1][1])
    image.rectangle((rect_start,rect_end),fill=fill,outline = color)
#Method to write the text inside the rectangle
def wr_text_in_rect(image,start_wr_w,start_wr_h,str,tfill):
    font = ImageFont.truetype("arial.ttf", 16)
    image.text((start_wr_w,start_wr_h),str, fill = tfill,font = font)
#Method to draw the top,test,env blocks (w.r.t. the value of n chosen)
def call_simple_rect(w,h,n,text,bfill,tfill,img1):
    w1 = w - (n*10); #end of x should be max
    if n != 5:
        h2 = h - (n*10); #end of 'y' should be max
        h1 = n*15 + 10;
        w2 = h1;
    elif n == 5: #The height of the env block should be less; So used like below dimensions
        h2 = h - (n*10*5)
        h1 = n*15 + 10 + 35;
        w2 = n*15 + 10;
    top_right = (w1,h1)
    bottom_left = (w2,h2)
    start_x = w1 - (50);
    start_y = h1 + (n*2);
    outline_width = 10
    outline_color = "black"
    draw_rect(img1,(top_right, bottom_left), fill=bfill ,color=outline_color, width=outline_width)
    wr_text_in_rect(img1,start_x,start_y,text,tfill)
    print ("Dimensions are %0d %0d %0d %0d",top_right, bottom_left)
    return w1;
#This ■docx■ module is to manipulate with docs like MS Word. Used it to add TB diagram to the document
import docx
#This ■opencv■ module in python ease us to draw arrowed line in the image
import cv2
# This ■pillow■ module to import Image draw module
from PIL import Image,ImageDraw,ImageFont
#Taking the handle ■doc■ for docx
doc = docx.Document()
#This Module is used to measure the whole screen size
```



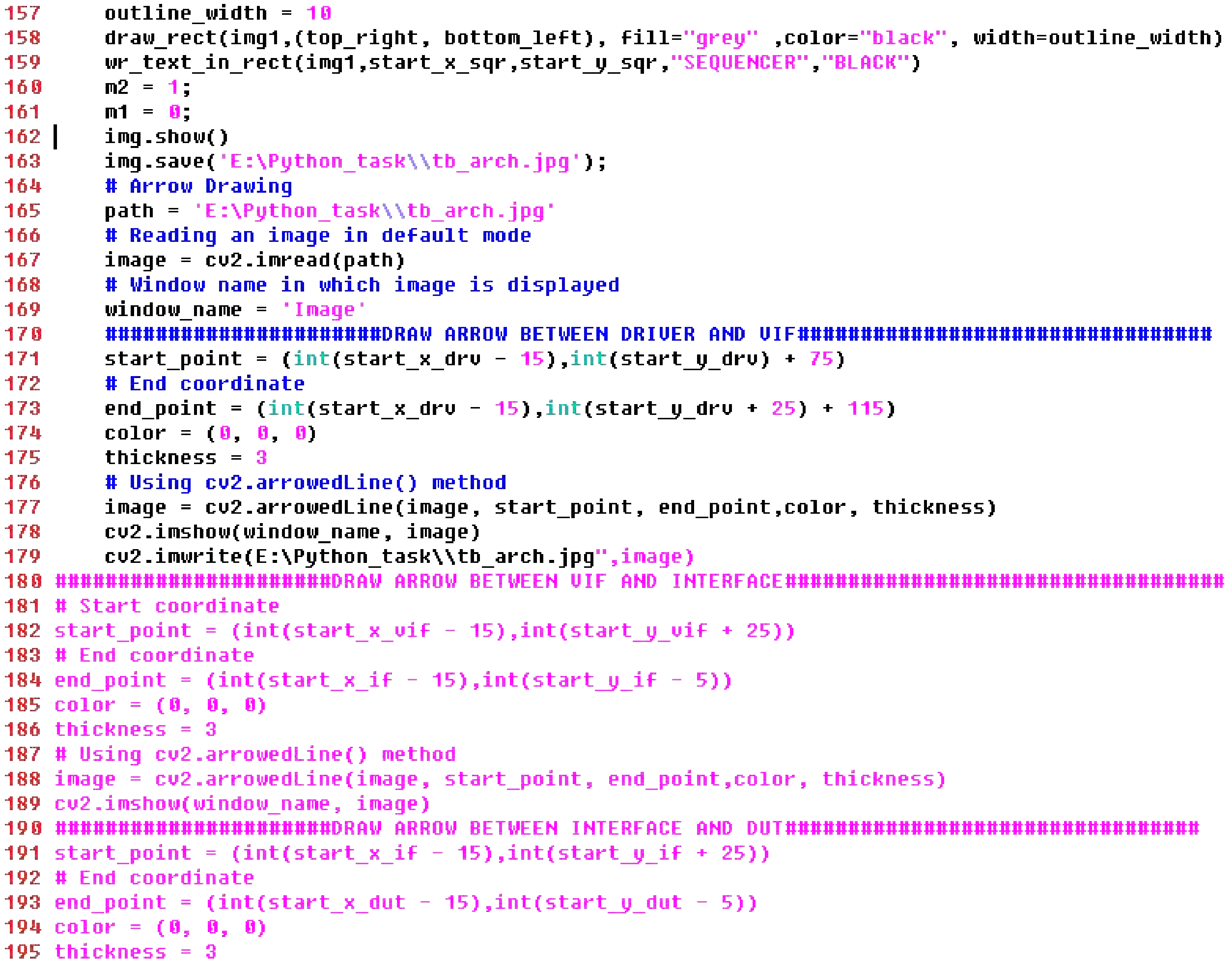
```
import tkinter
root = tkinter.Tk()
width = root.winfo_screenwidth()
height = root.winfo_screenheight()
print ("Width & HEIGHT",width,height)
# create line image of width and height
w = width
h = height
img = Image.new("RGB", (w, h),"white")
img1 = ImageDraw.Draw(img)
#Create first outer rectangle top n=1
n = 1;
top_dim = call_simple_rect(w,h,n,"TOP","orange","black",img1);
#Create second inner rectangle test n=3
n = 3;
test_dim = call_simple_rect(w,h,n,"TEST","pink","black",img1);
#Create third inner rectangle env n=5
n = 5;
env_dim = call_simple_rect(w,h,n,"ENV","yellow","black",img1);
#Create fifth inner rectangle SCOREBOARD
top_right = (w-140,150)
bottom_left = (400,230)
start_x = ((w-140)+400)/2 + 12;
start_y = 180;
draw_rect(img1,(top_right, bottom_left), fill="gray" ,color="black", width=10)
wr_text_in_rect(img1,start_x,start_y,"SCOREBOARD","BLACK")
#Create fourth inner rectangle sequences DUT
top_right = (90,h-80)
bottom_left = (w-40,h-50)
start_x_dut = (90 + (w-40))/2;
start_y_dut = (((h - 80) + (h - 50))/2) - 12;
draw_rect(img1,(top_right, bottom_left), fill="grey" ,color="black", width=10)
wr_text_in_rect(img1,start_x_dut,start_y_dut,"DUT","BLACK")
#Create fourth inner rectangle Interface
top_right = (90,h-120)
bottom_left = (w-40,h-150)
start_x_if = (90 + (w-40))/2;
start_y_if = ((h-120)+(h-150))/2 - 12;
draw_rect(img1,(top_right, bottom_left), fill="green" ,color="black", width=10)
```



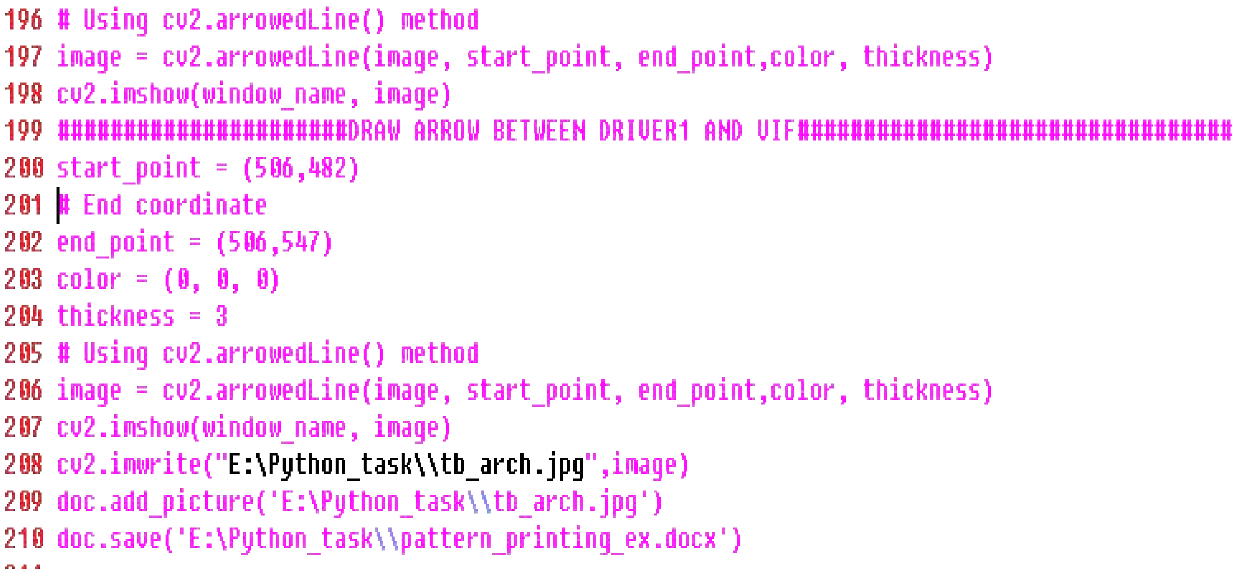
```
wr_text_in_rect(img1,start_x_if,start_y_if,"INTERFACE","BLACK")
#Create fourth inner rectangle VIF
top_right = (90,h-190)
bottom_left = (w-40,h-220)
start_x_vif = (90 + (w-40))/2;
start_y_vif = ((h-190)+(h-220))/2 - 12;
draw_rect(img1,(top_right, bottom_left), fill="gray" ,color="black", width=10)
wr_text_in_rect(img1,start_x_vif,start_y_vif,"VIF","BLACK")
print(env_dim);
#Check for number of agents
n = 7;
agnt_cnt = int(input("Enter no. of agents"))
w1 = (env_dim/agnt_cnt)
h1 = (env_dim/agnt_cnt)
tx = 40;
x0 = 0;
diff = 0;
m1 = 1;
m2 = 0;
#To draw the number of agents w.r.t. agent count
for val in range(agnt_cnt):
    print("VALUE OF X0 IS",x0)
    x1 = tx + 55;
    print("VALUE OF X1 IS",x1)
    y0 = (n*4*10)
    if agnt_cnt == 1:
        x0 = (env_dim/agnt_cnt) - 20;
    elif agnt_cnt != 1:
        x0 = (m1*(env_dim/agnt_cnt))+(m2*(x1 + diff));
    print("VALUE OF X0 LATER IS",x0)
    y1 = h - (4*n*10)
    tx = x0;
    diff = x0 - x1;
    top_right = (x0,y0);
    bottom_left = (x1,y1);
    start_x = x0 - 50;
    start_y = y0 + 5;
    outline_width = 10
    print(top_right);
```



```
    print(bottom_left);
    draw_rect(img1,(top_right, bottom_left), fill="cyan" ,color="black", width=outline_width)
    wr_text_in_rect(img1,start_x,start_y,"AGENT","BLACK")
    #Start another rectangle MONITOR inside the agent
    x3 = x1 + 10
    y3 = y0 + 20
    x2 = x0 - 350
    y2 = y0 + 100
    top_right = (x2,y2);
    bottom_left = (x3,y3);
    start_x_mon = (x2 + x3)/2;
    start_y_mon = y3 + 15;
    outline_width = 10
    draw_rect(img1,(top_right, bottom_left), fill="orange" ,color="black", width=outline_width)
    wr_text_in_rect(img1,start_x_mon,start_y_mon,"MONITOR","BLACK")
    #Start another rectangle DRIVER inside the agent
    x5 = x3 + 300
    y5 = y3 + 100
    x4 = x2 + 300
    y4 = y2 + 100
    top_right = (x4,y4);
    bottom_left = (x5,y5);
    start_x_drv = (x4 + x5)/2;
    start_y_drv = y5 + 5;
    outline_width = 10
    draw_rect(img1,(top_right, bottom_left), fill="green" ,color="black", width=outline_width)
    wr_text_in_rect(img1,start_x_drv,start_y_drv,"DRIVER","BLACK")
    m2 = 1;
    m1 = 0;
    print("DRIVER",start_x_drv,start_y_drv)
    #Start another rectangle SEQUENCER inside the agent
    x5 = x3 + 300
    y5 = y3
    x4 = x2 + 300
    y4 = y2
    top_right = (x4,y4);
    bottom_left = (x5,y5);
    start_x_sqr = (x4 + x5)/2;
    start_y_sqr = y5 + 5;
```

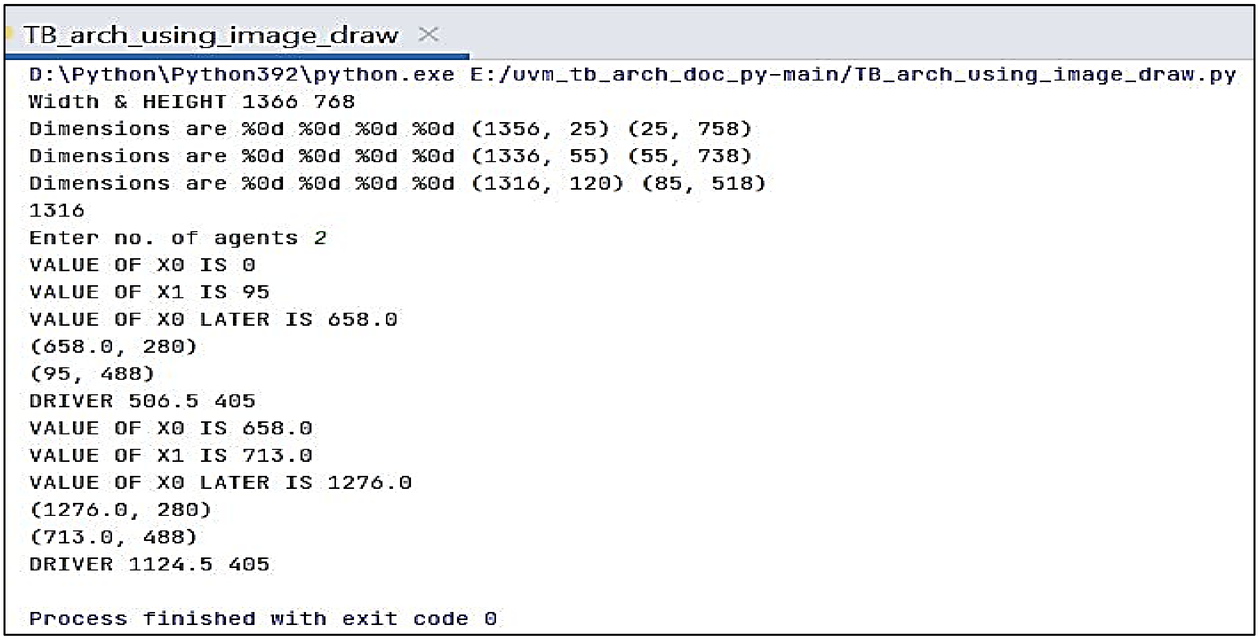


```
    outline_width = 10
    draw_rect(img1,(top_right, bottom_left), fill="grey" ,color="black", width=outline_width)
    wr_text_in_rect(img1,start_x_sqr,start_y_sqr,"SEQUENCER","BLACK")
    m2 = 1;
    m1 = 0;
    img.show()
    img.save('E:\Python_task\\tb_arch.jpg');
    # Arrow Drawing
    path = 'E:\Python_task\\tb_arch.jpg'
    # Reading an image in default mode
    image = cv2.imread(path)
    # Window name in which image is displayed
    window_name = 'Image'
    ######################DRAW ARROW BETWEEN DRIVER AND VIF#################################
    start_point = (int(start_x_drv - 15),int(start_y_drv) + 75)
    # End coordinate
    end_point = (int(start_x_drv - 15),int(start_y_drv + 25) + 115)
    color = (0, 0, 0)
    thickness = 3
    # Using cv2.arrowedLine() method
    image = cv2.arrowedLine(image, start_point, end_point,color, thickness)
    cv2.imshow(window_name, image)
    cv2.imwrite(E:\Python_task\\tb_arch.jpg",image)
######################DRAW ARROW BETWEEN VIF AND INTERFACE##################################
# Start coordinate
start_point = (int(start_x_vif - 15),int(start_y_vif + 25))
# End coordinate
end_point = (int(start_x_if - 15),int(start_y_if - 5))
color = (0, 0, 0)
thickness = 3
# Using cv2.arrowedLine() method
image = cv2.arrowedLine(image, start_point, end_point,color, thickness)
cv2.imshow(window_name, image)
######################DRAW ARROW BETWEEN INTERFACE AND DUT#################################
start_point = (int(start_x_if - 15),int(start_y_if + 25))
# End coordinate
end_point = (int(start_x_dut - 15),int(start_y_dut - 5))
color = (0, 0, 0)
thickness = 3
```

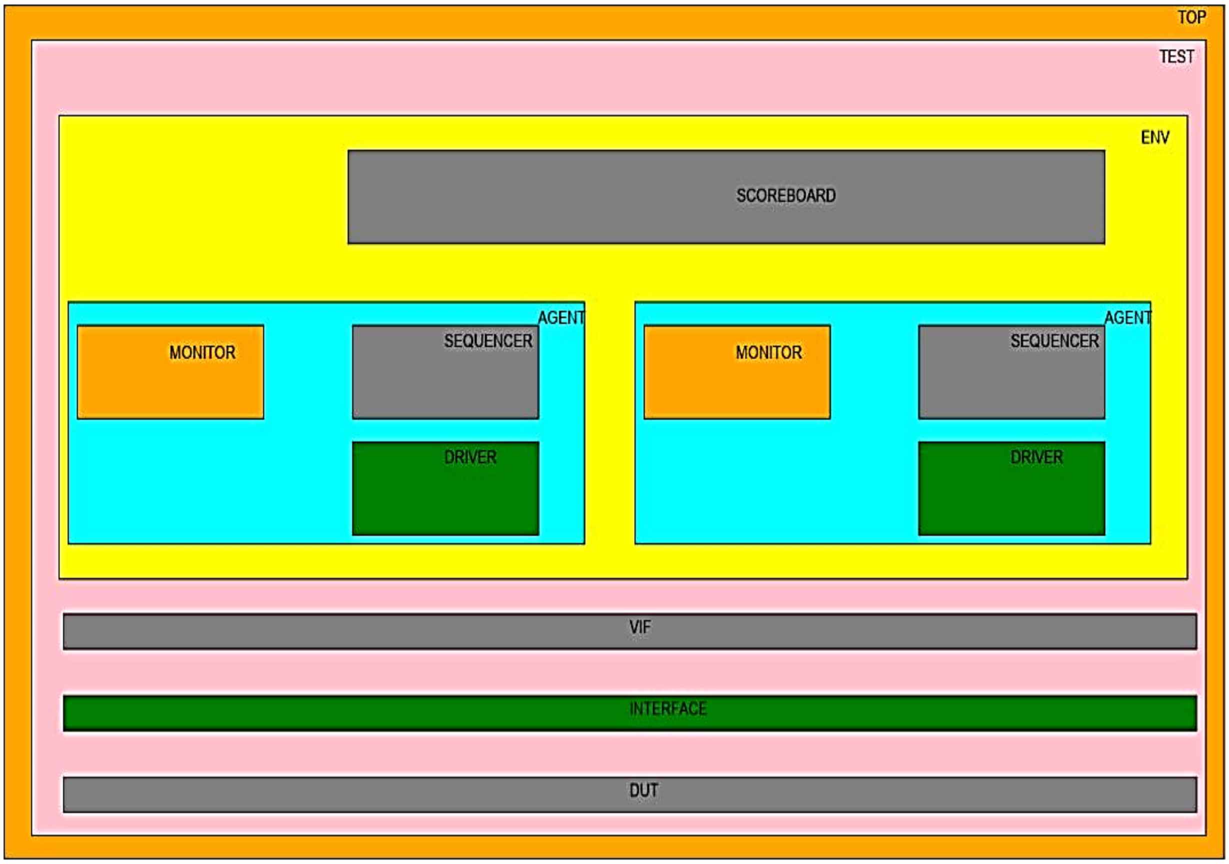


```
# Using cv2.arrowedLine() method
image = cv2.arrowedLine(image, start_point, end_point,color, thickness)
cv2.imshow(window_name, image)
######################DRAW ARROW BETWEEN DRIVER1 AND VIF################################
start_point = (506,482)
# End coordinate
end_point = (506,547)
color = (0, 0, 0)
thickness = 3
# Using cv2.arrowedLine() method
image = cv2.arrowedLine(image, start_point, end_point,color, thickness)
cv2.imshow(window_name, image)
cv2.imwrite("E:\Python_task\\tb_arch.jpg",image)
doc.add_picture('E:\Python_task\\tb_arch.jpg')
doc.save('E:\Python_task\\pattern_printing_ex.docx')
```

Results of TB_arch_using_image_draw script



```
D:\Python\Python392\python.exe E:/uvm_tb_arch_doc_py-main/TB_arch_using_image_draw.py
Width & HEIGHT 1366 768
Dimensions are %0d %0d %0d %0d (1356, 25) (25, 758)
Dimensions are %0d %0d %0d %0d (1336, 55) (55, 738)
Dimensions are %0d %0d %0d %0d (1316, 120) (85, 518)
1316
Enter no. of agents 2
VALUE OF X0 IS 0
VALUE OF X1 IS 95
VALUE OF X0 LATER IS 658.0
(658.0, 280)
(95, 488)
DRIVER 506.5 405
VALUE OF X0 IS 658.0
VALUE OF X1 IS 713.0
VALUE OF X0 LATER IS 1276.0
(1276.0, 280)
(713.0, 488)
DRIVER 1124.5 405

Process finished with exit code 0
```

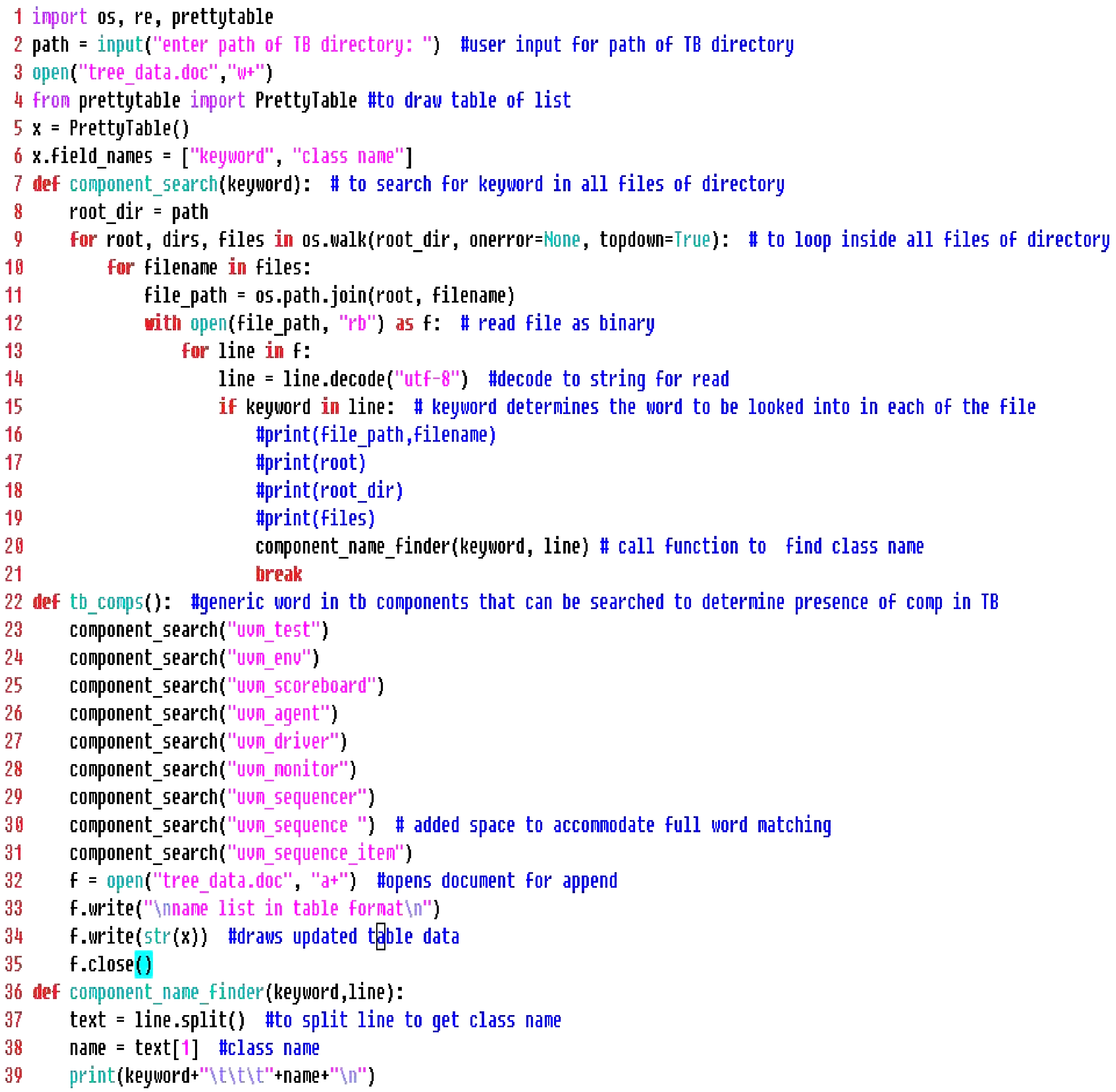


3.2 component_name_finder.py

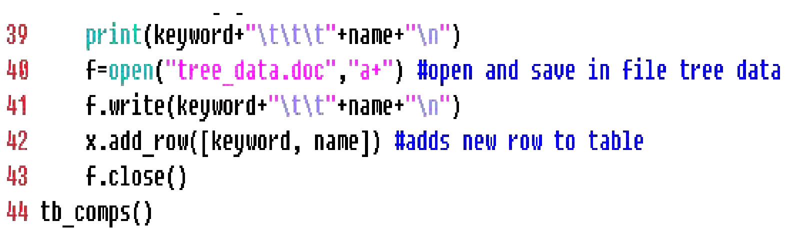
The component name finder script checks the particular keyword in all files all lines and collects the class name of components here user have to give the input directory path then it prints the output in normal and table format for further processing

Import the os, re, prettytable modules enter the testbench directory path, Open the doc file and then import the pretty table and assign the Class names of the Testbench code as a Keyword

Then write the function for component search and then assign the Input directory path to root directory path. Here we have to call the component search function and checking the keyword of TB class names and opening the docx file write the information in the file and closing the file

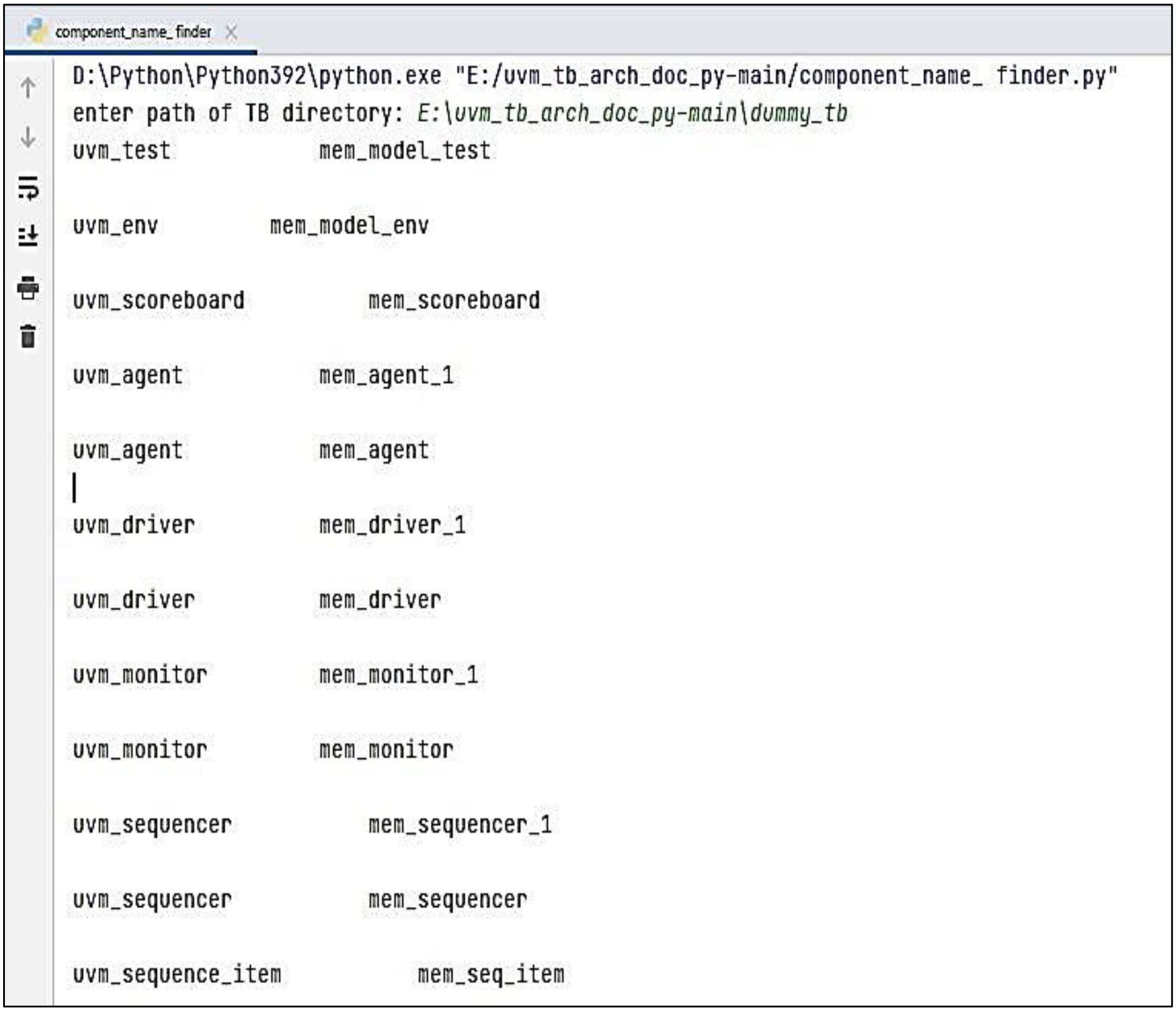


```python
import os, re, prettytable
path = input("enter path of TB directory: ")  #user input for path of TB directory
open("tree_data.doc","w+")
from prettytable import PrettyTable #to draw table of list
x = PrettyTable()
x.field_names = ["keyword", "class name"]
def component_search(keyword):  # to search for keyword in all files of directory
    root_dir = path
    for root, dirs, files in os.walk(root_dir, onerror=None, topdown=True):  # to loop inside all files of directory
        for filename in files:
            file_path = os.path.join(root, filename)
            with open(file_path, "rb") as f:  # read file as binary
                for line in f:
                    line = line.decode("utf-8")  #decode to string for read
                    if keyword in line:  # keyword determines the word to be looked into in each of the file
                        #print(file_path,filename)
                        #print(root)
                        #print(root_dir)
                        #print(files)
                        component_name_finder(keyword, line) # call function to  find class name
                        break
def tb_comps():  #generic word in tb components that can be searched to determine presence of comp in TB
    component_search("uvm_test")
    component_search("uvm_env")
    component_search("uvm_scoreboard")
    component_search("uvm_agent")
    component_search("uvm_driver")
    component_search("uvm_monitor")
    component_search("uvm_sequencer")
    component_search("uvm_sequence ")  # added space to accommodate full word matching
    component_search("uvm_sequence_item")
    f = open("tree_data.doc", "a+")  #opens document for append
    f.write("\nname list in table format\n")
    f.write(str(x))  #draws updated table data
    f.close()
def component_name_finder(keyword,line):
    text = line.split()  #to split line to get class name
    name = text[1]  #class name
    print(keyword+"\t\t\t"+name+"\n")
```



```python
    print(keyword+"\t\t\t"+name+"\n")
    f=open("tree_data.doc","a+") #open and save in file tree data
    f.write(keyword+"\t\t"+name+"\n")
    x.add_row([keyword, name]) #adds new row to table
    f.close()
tb_comps()
```

Results of component_name_finder.py script



```
D:\Python\Python392\python.exe "E:/uvm_tb_arch_doc_py-main/component_name_ finder.py"
enter path of TB directory: E:\uvm_tb_arch_doc_py-main\dummy_tb
uvm_test            mem_model_test


uvm_env         mem_model_env


uvm_scoreboard          mem_scoreboard


uvm_agent           mem_agent_1


uvm_agent           mem_agent


uvm_driver          mem_driver_1


uvm_driver          mem_driver


uvm_monitor         mem_monitor_1


uvm_monitor         mem_monitor


uvm_sequencer           mem_sequencer_1


uvm_sequencer           mem_sequencer


uvm_sequence_item           mem_seq_item
```

3.3 draw_rect_using_turtle.py

1)To draw the shapes using Turtle, the major shapes that are needed to construct the Testbench architecture are square and rectangle. Below is the Python script to draw square and rectangle and also connectivity

Steps:

1.Import the turtle library

2.Set the Screen color

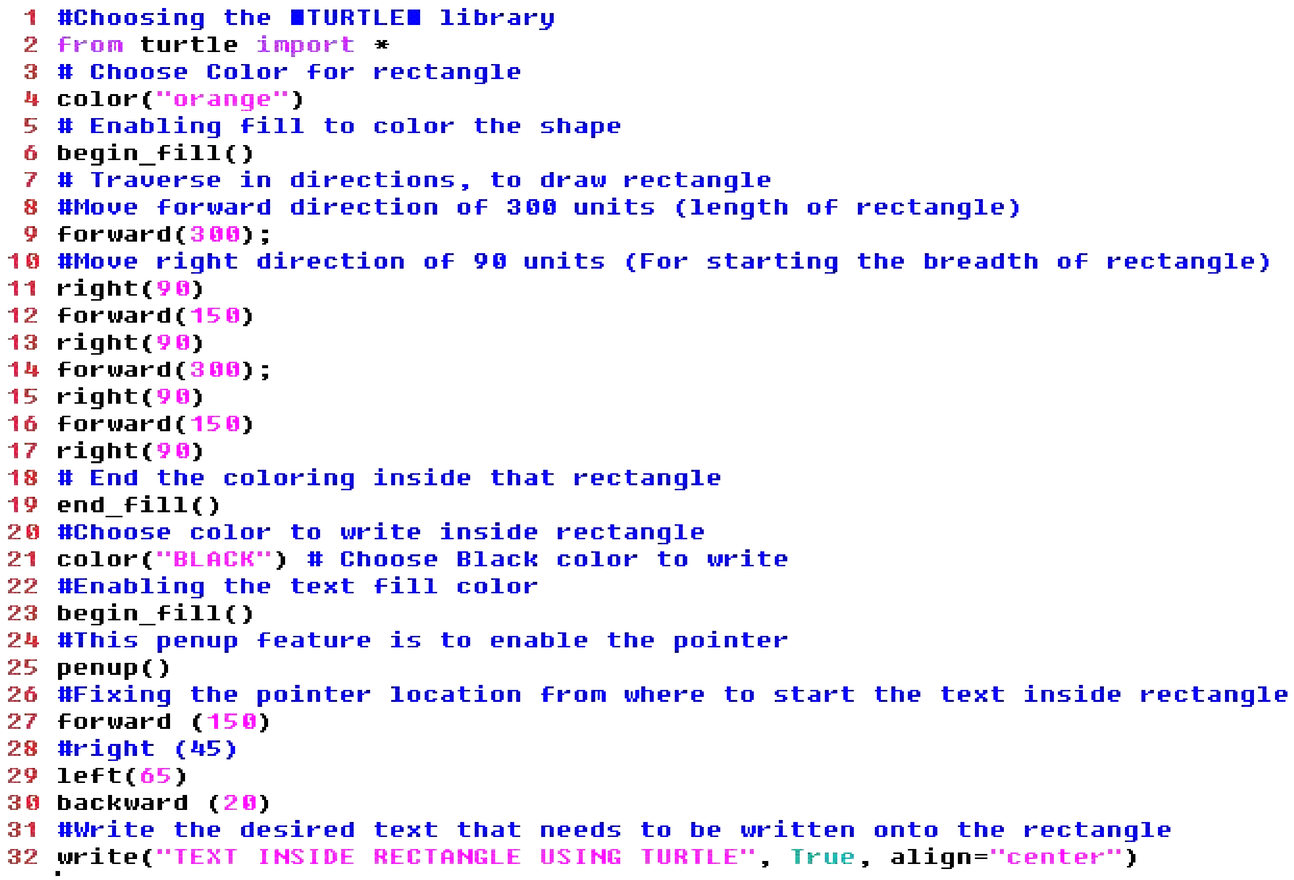
3.Instantiate the object for Turtle

4.Set the pen color

5.Define function draw_square and draw_rect. Pass the co-ordinates,length,width and color as the input to the function

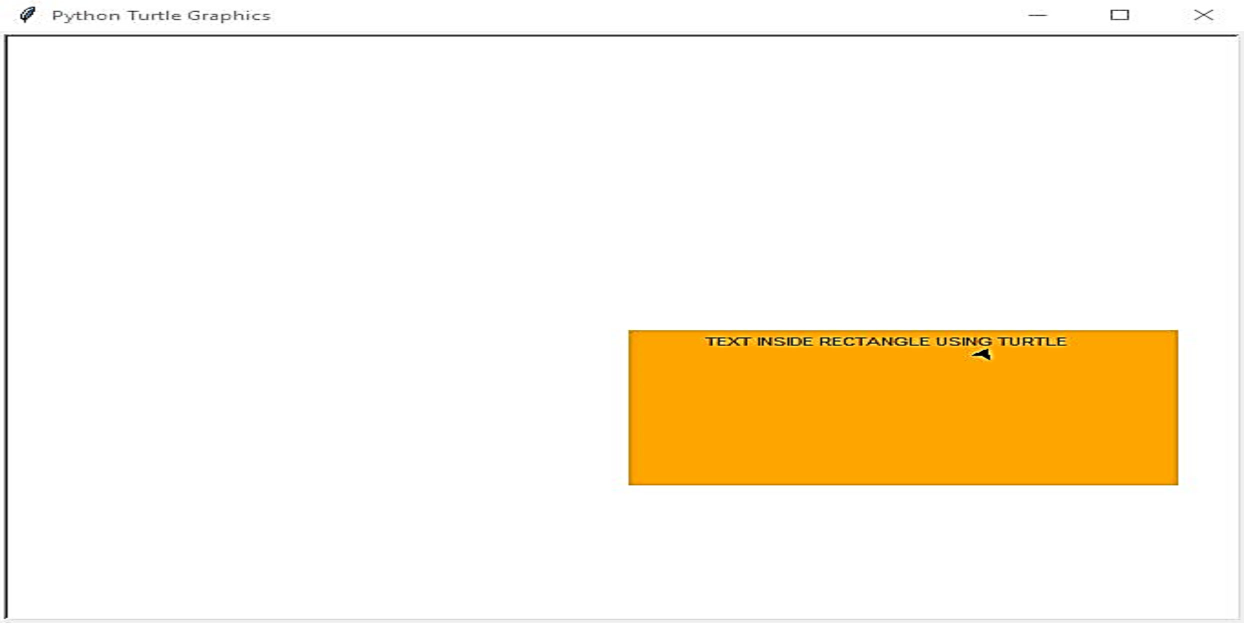
6.Move to the desired location(co-ordinate) to draw the shape

7.Use begin_fill and end_fill to fill in the shape



```
#Choosing the ■TURTLE■ library
from turtle import *
# Choose Color for rectangle
color("orange")
# Enabling fill to color the shape
begin_fill()
# Traverse in directions, to draw rectangle
#Move forward direction of 300 units (length of rectangle)
forward(300);
#Move right direction of 90 units (For starting the breadth of rectangle)
right(90)
forward(150)
right(90)
forward(300);
right(90)
forward(150)
right(90)
# End the coloring inside that rectangle
end_fill()
#Choose color to write inside rectangle
color("BLACK") # Choose Black color to write
#Enabling the text fill color
begin_fill()
#This penup feature is to enable the pointer
penup()
#Fixing the pointer location from where to start the text inside rectangle
forward (150)
#right (45)
left(65)
backward (20)
#Write the desired text that needs to be written onto the rectangle
write("TEXT INSIDE RECTANGLE USING TURTLE", True, align="center")
```

Results of draw_rect_using_turtle.py script

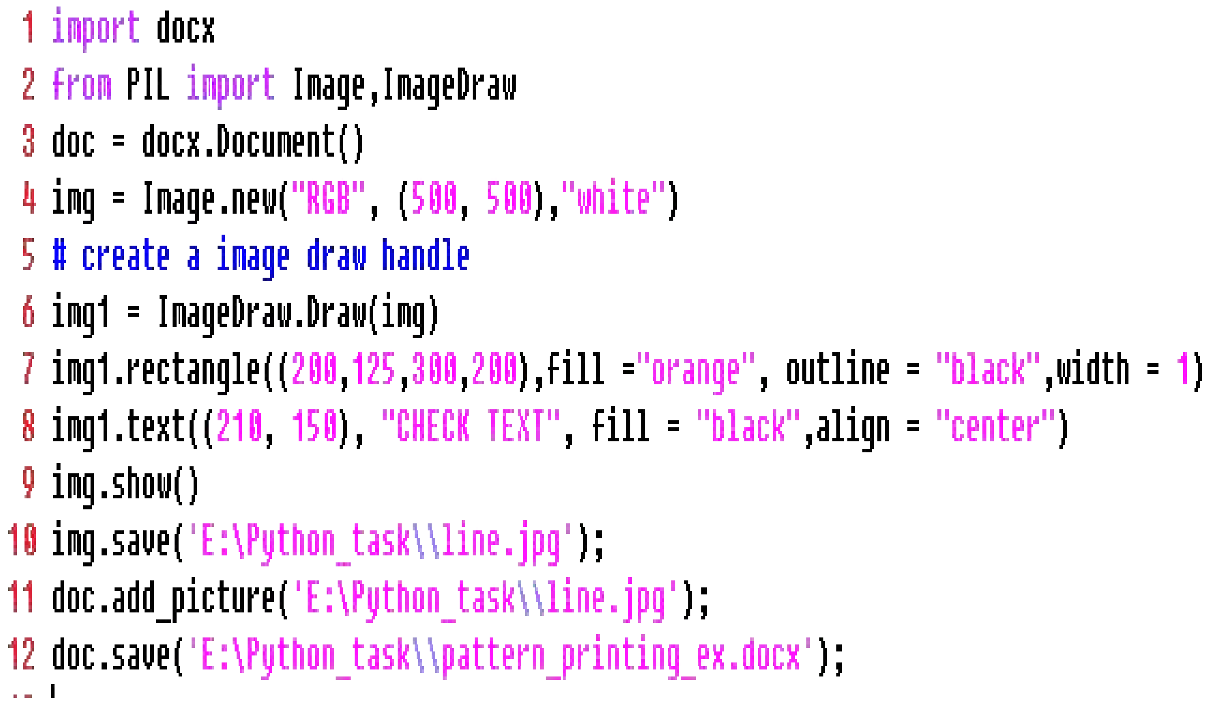


3.4 draw_rect_with_text_inside.py(using Image draw module)

To draw the shapes using Image draw module, the major shapes that are needed to construct the Testbench architecture are square and rectangle. Below is the Python script to draw square and rectangle and also connectivity

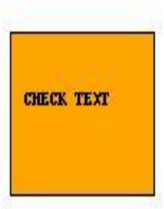
Steps:

- Created an empty image *.jpg file
- Drew a rectangle in that image using ImageDraw module
- Saved the image & using the Image Draw module, inserted the text inside the rectangle (by changing the required dimensions in trial and error manner)
- Added that .jpg file into a .docx document & saved that. (using docx module)



```
import docx
from PIL import Image,ImageDraw
doc = docx.Document()
img = Image.new("RGB", (500, 500),"white")
# create a image draw handle
img1 = ImageDraw.Draw(img)
img1.rectangle((200,125,300,200),fill ="orange", outline = "black",width = 1)
img1.text((210, 150), "CHECK TEXT", fill = "black",align = "center")
img.show()
img.save('E:\Python_task\\line.jpg');
doc.add_picture('E:\Python_task\\line.jpg');
doc.save('E:\Python_task\\pattern_printing_ex.docx');
```

Results of draw_rect_with_text_inside.py script(using Image draw module)



3.6 tb_arch_img_draw.py

To read the file which contains the information about
Testbench and to draw the agent components

Steps:

1. Read the file tb_info.txt which contains the agent information (Number of agent). Read the first line and then using split method the words are stored in a list form which the number can be retrieved.
2. Define the method top,test,env, agent, scoreboard, driver, sequencer and monitor to draw the corresponding component
3. In the method definition, use the in-built function called rectangle from the ImageDraw module to draw the components
4. The arguments to the functions are:

   x0,y0 – Starting co-ordinate to draw the rectangle

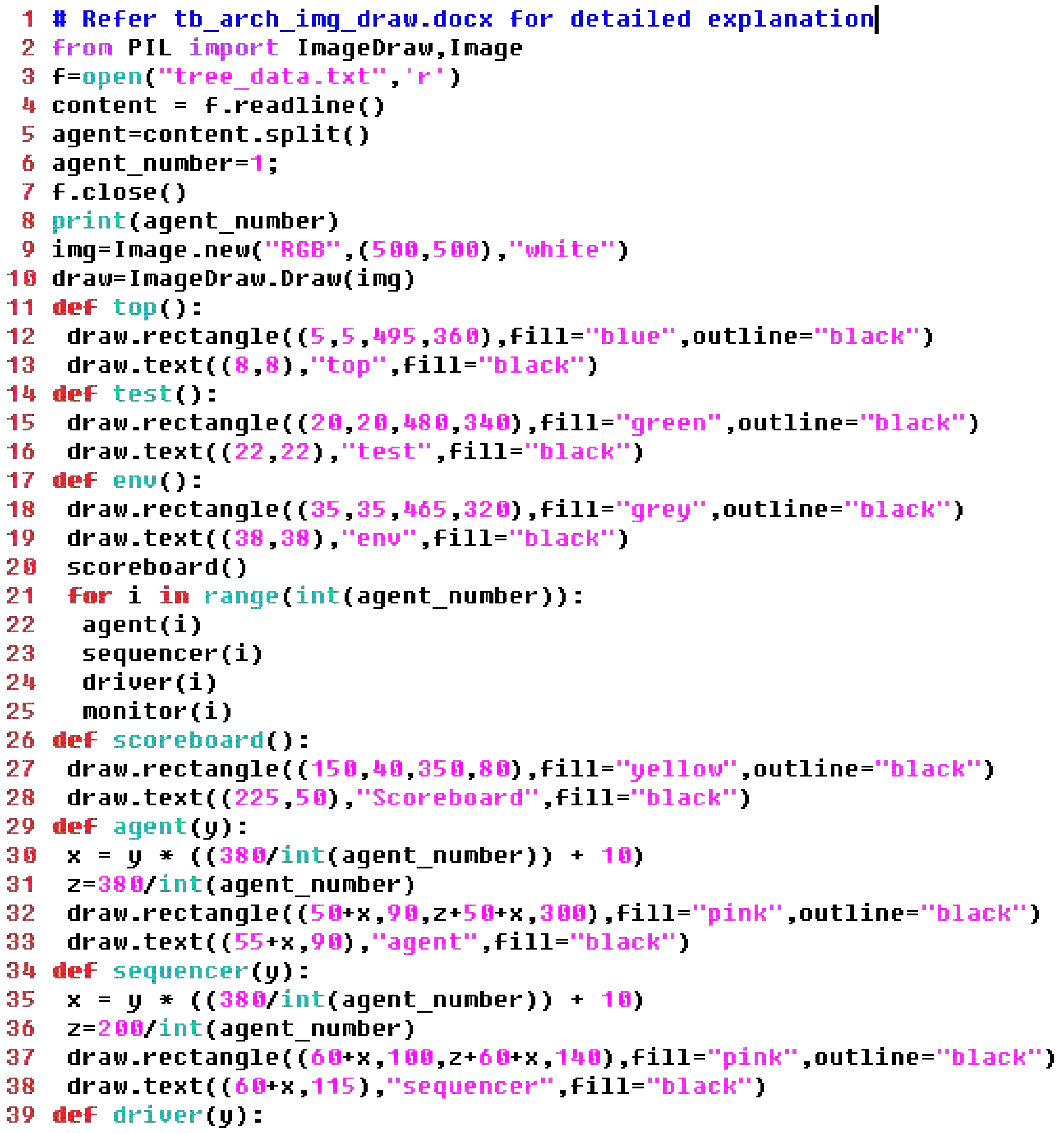
   x1,y1 - End co-ordinate

   Outline – Outline color for the shape

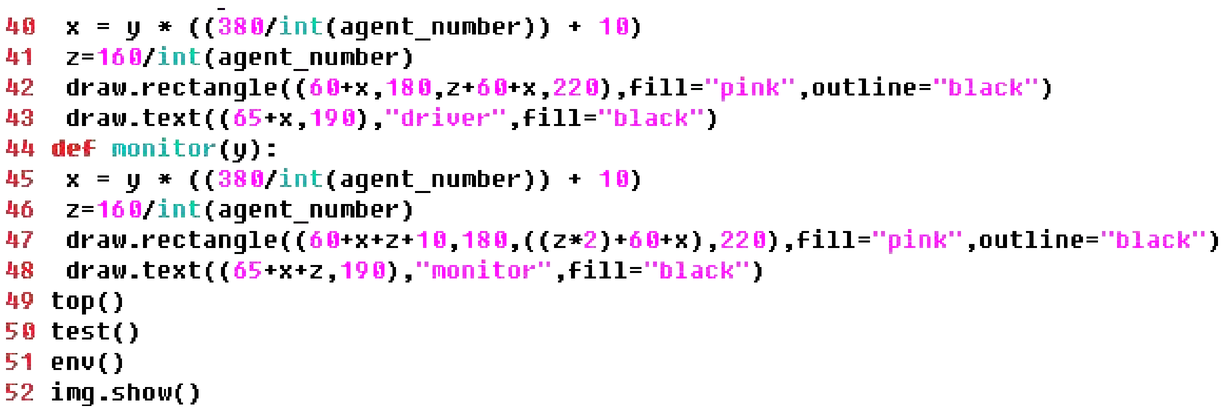
   1.Co-ordinates – where the text needs to be inserted

   2.Text in the form of string

   3.Color for the text
5. .Agents and its components are constructed based on the agent numbers. From env() the scoreboard() and agent components are called
6. The text method in Image Draw module issued to include the text. The parameters are: Coordinates, text, clour
7. The information collected from the testbench are written in a file. Here the tb_info.txt file contains agent information (number of agents)

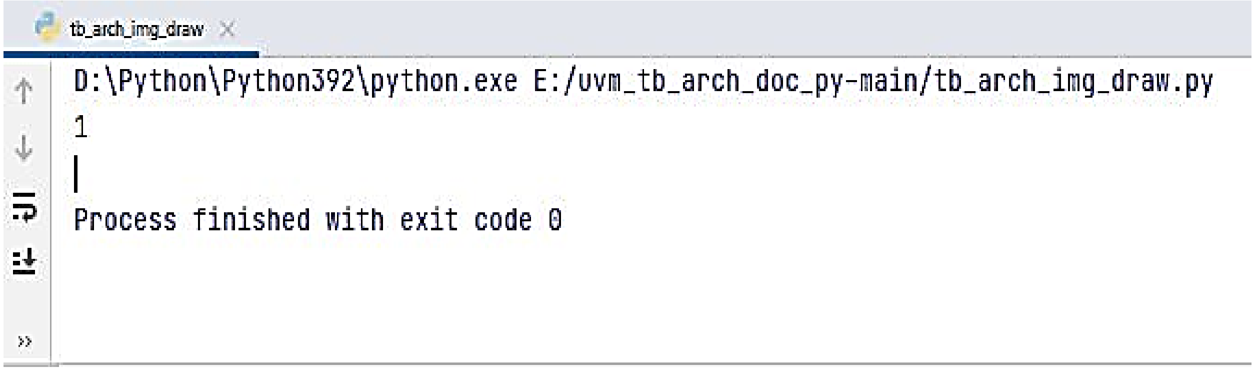


```
# Refer tb_arch_img_draw.docx for detailed explanation
from PIL import ImageDraw,Image
f=open("tree_data.txt",'r')
content = f.readline()
agent=content.split()
agent_number=1;
f.close()
print(agent_number)
img=Image.new("RGB",(500,500),"white")
draw=ImageDraw.Draw(img)
def top():
 draw.rectangle((5,5,495,360),fill="blue",outline="black")
 draw.text((8,8),"top",fill="black")
def test():
 draw.rectangle((20,20,480,340),fill="green",outline="black")
 draw.text((22,22),"test",fill="black")
def env():
 draw.rectangle((35,35,465,320),fill="grey",outline="black")
 draw.text((38,38),"env",fill="black")
 scoreboard()
 for i in range(int(agent_number)):
  agent(i)
  sequencer(i)
  driver(i)
  monitor(i)
def scoreboard():
 draw.rectangle((150,40,350,80),fill="yellow",outline="black")
 draw.text((225,50),"Scoreboard",fill="black")
def agent(y):
 x = y * ((380/int(agent_number)) + 10)
 z=380/int(agent_number)
 draw.rectangle((50+x,90,z+50+x,300),fill="pink",outline="black")
 draw.text((55+x,90),"agent",fill="black")
def sequencer(y):
 x = y * ((380/int(agent_number)) + 10)
 z=200/int(agent_number)
 draw.rectangle((60+x,100,z+60+x,140),fill="pink",outline="black")
 draw.text((60+x,115),"sequencer",fill="black")
def driver(y):
```

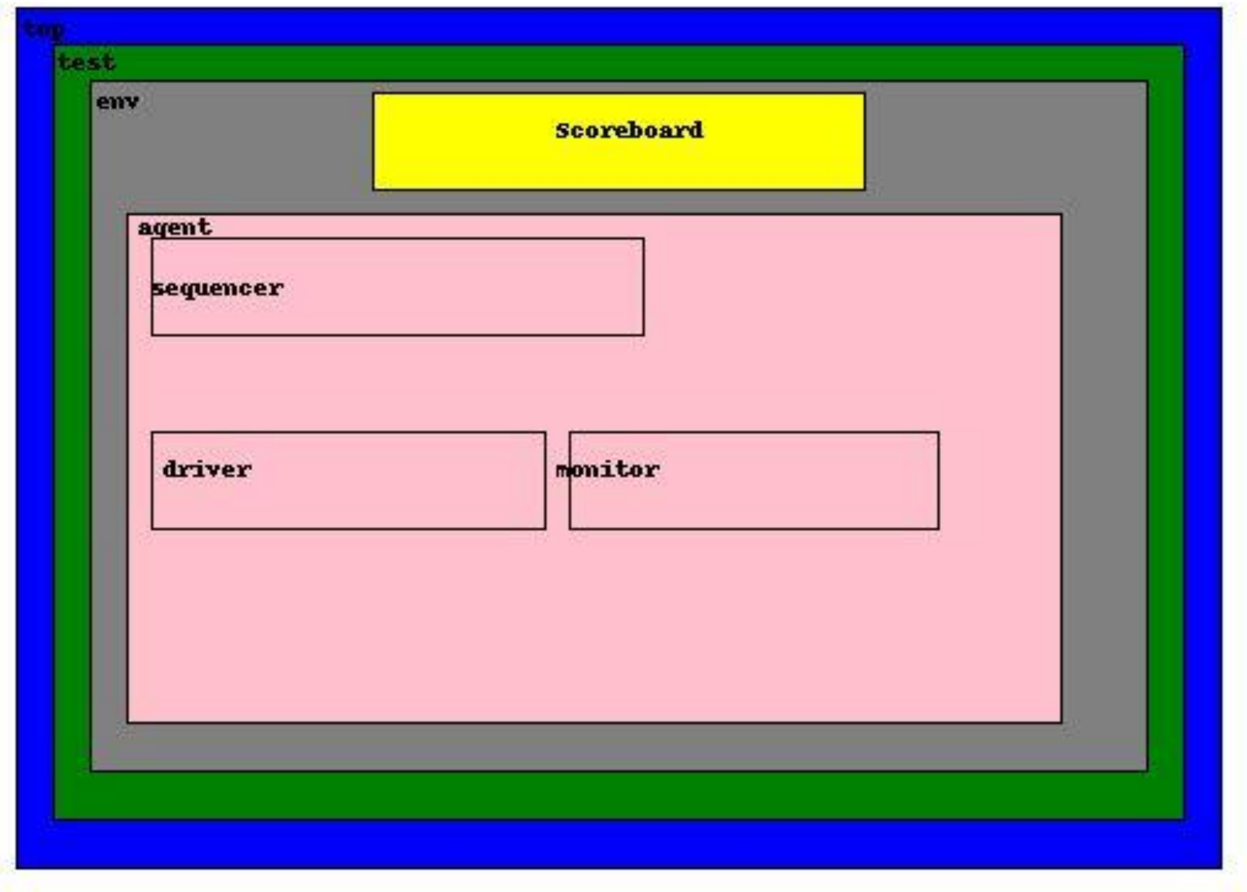


```
 x = y * ((380/int(agent_number)) + 10)
 z=160/int(agent_number)
 draw.rectangle((60+x,180,z+60+x,220),fill="pink",outline="black")
 draw.text((65+x,190),"driver",fill="black")
def monitor(y):
 x = y * ((380/int(agent_number)) + 10)
 z=160/int(agent_number)
 draw.rectangle((60+x+z+10,180,((z*2)+60+x),220),fill="pink",outline="black")
 draw.text((65+x+z,190),"monitor",fill="black")
top()
test()
env()
img.show()
```

Results of tb_arch_img_draw.py script



```
D:\Python\Python392\python.exe E:/uvm_tb_arch_doc_py-main/tb_arch_img_draw.py
1

Process finished with exit code 0
```



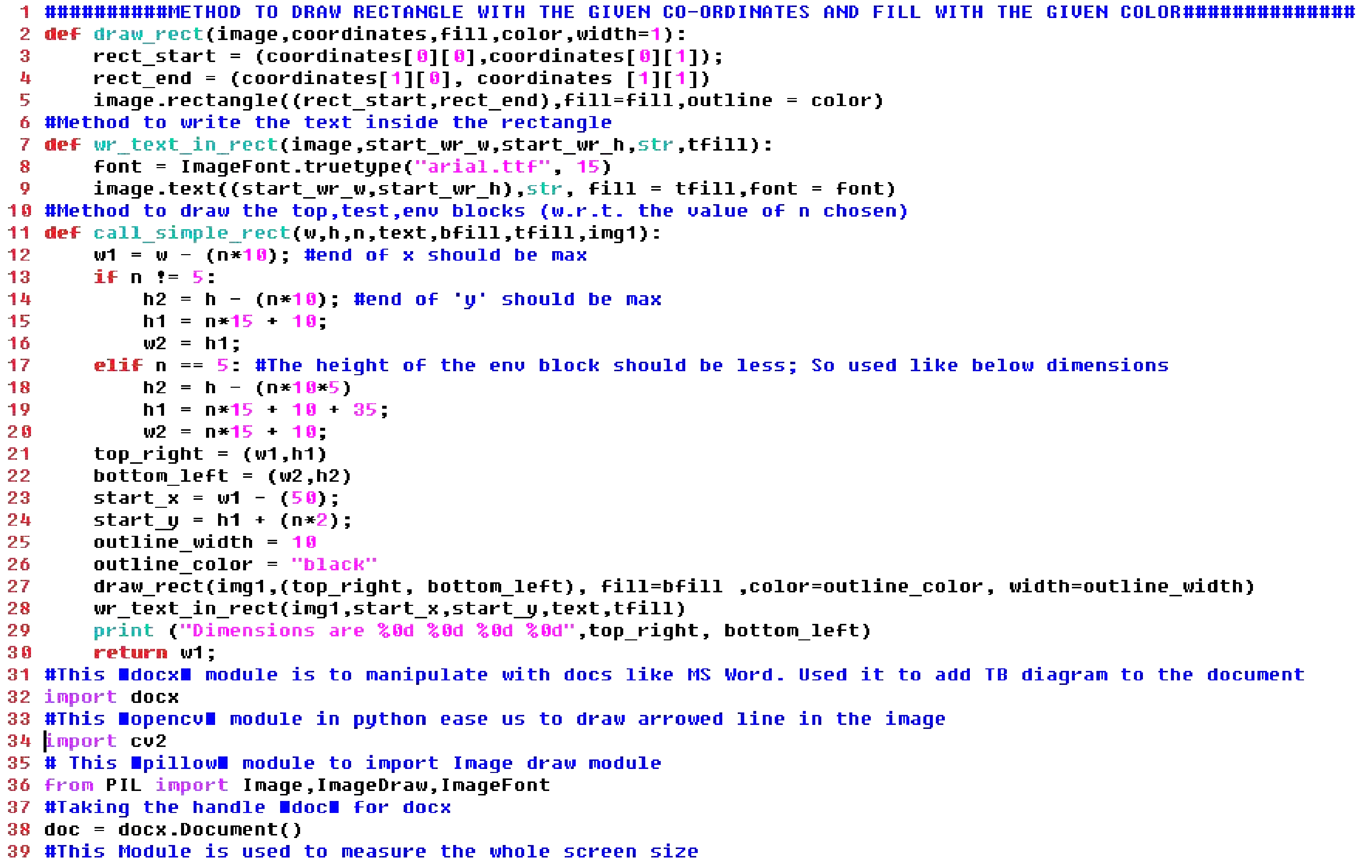
3.7 uvm_tb_arch_agent.py

- To draw TB Diagram aimed at 2 agentsTo draw Rectangle with given co-ordinate & fill with given colour and write the text inside rectangle

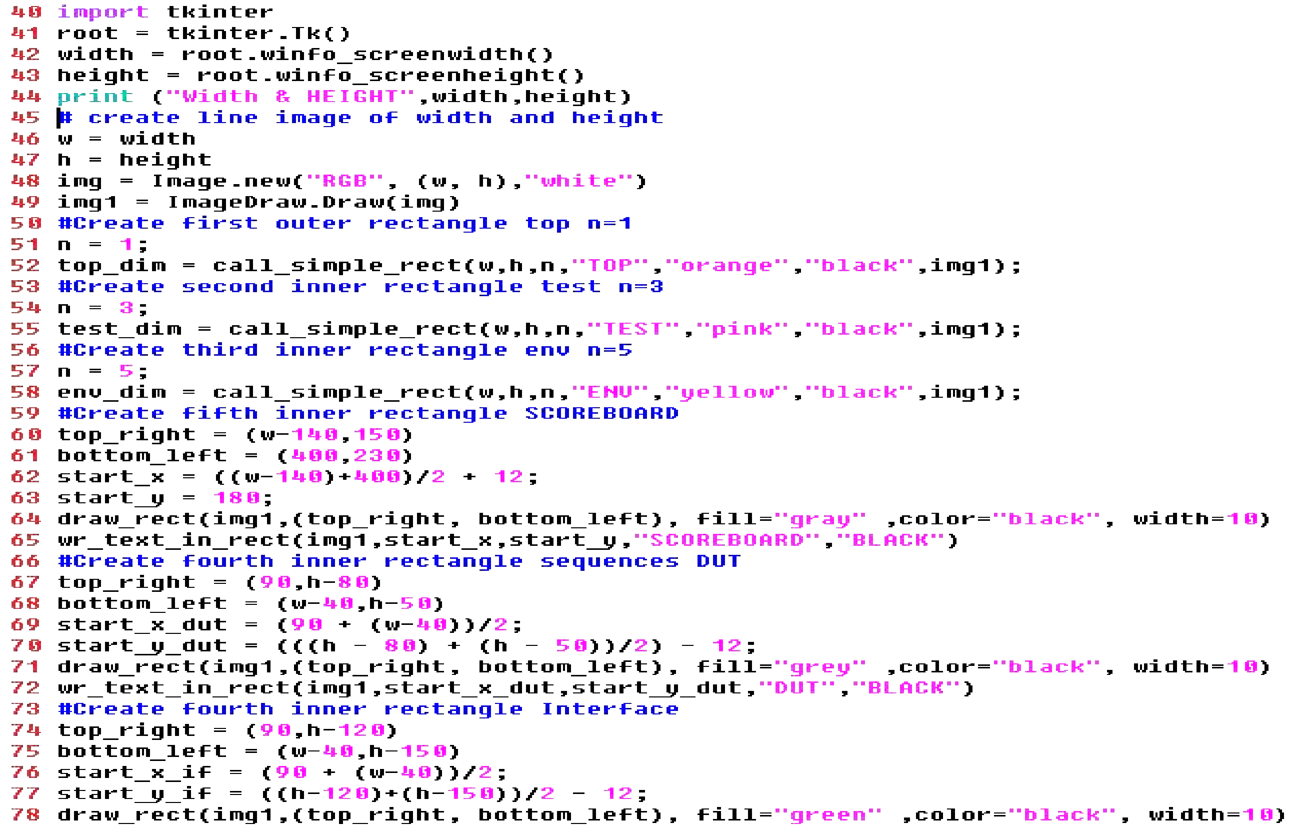
- To draw top, test, env blocks write the value of n choosen height of env block should be less , so we have to give proper dimensions
- This “docx” module is to manipulate with docs like MS Word. Used it to add TB diagram to the document, we have take handle doc for docx

- After we have to import tkinter & setting height & width to measure whole screen size and then
    - Create 1st outer rectangle for top
    - Create 2nd inner rectangle for test
    - Create 3rd inner rectangle for env
    - Create 5th inner rectangle for scoreboard
    - Create 4th inner rectangle for sequences, DUT, Interface, Virtual Interface
    - Check for No of agents user have to give
    - Start another rectangle MON inside agent
    - Start another rectangle DRV inside agent
    - Start another rectangle SEQR inside agent

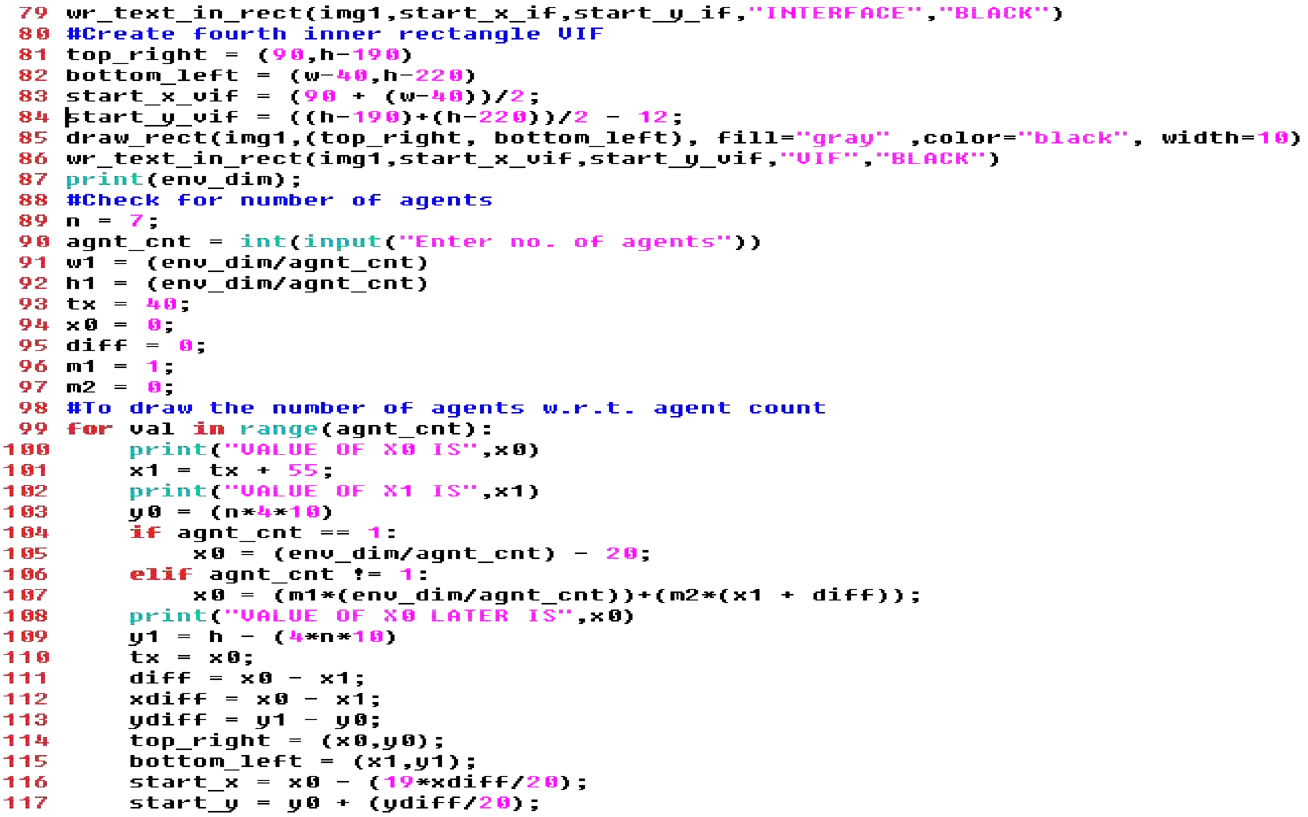
- Then draw the arrows between Driver to Virtual Interface, Virtual Interface to Interface and Driver1 to Virtual Interface and then set the start & end co-ordinates
- By using arrowed line() method set the colour & thickness and then add the picture & save the picture in docx



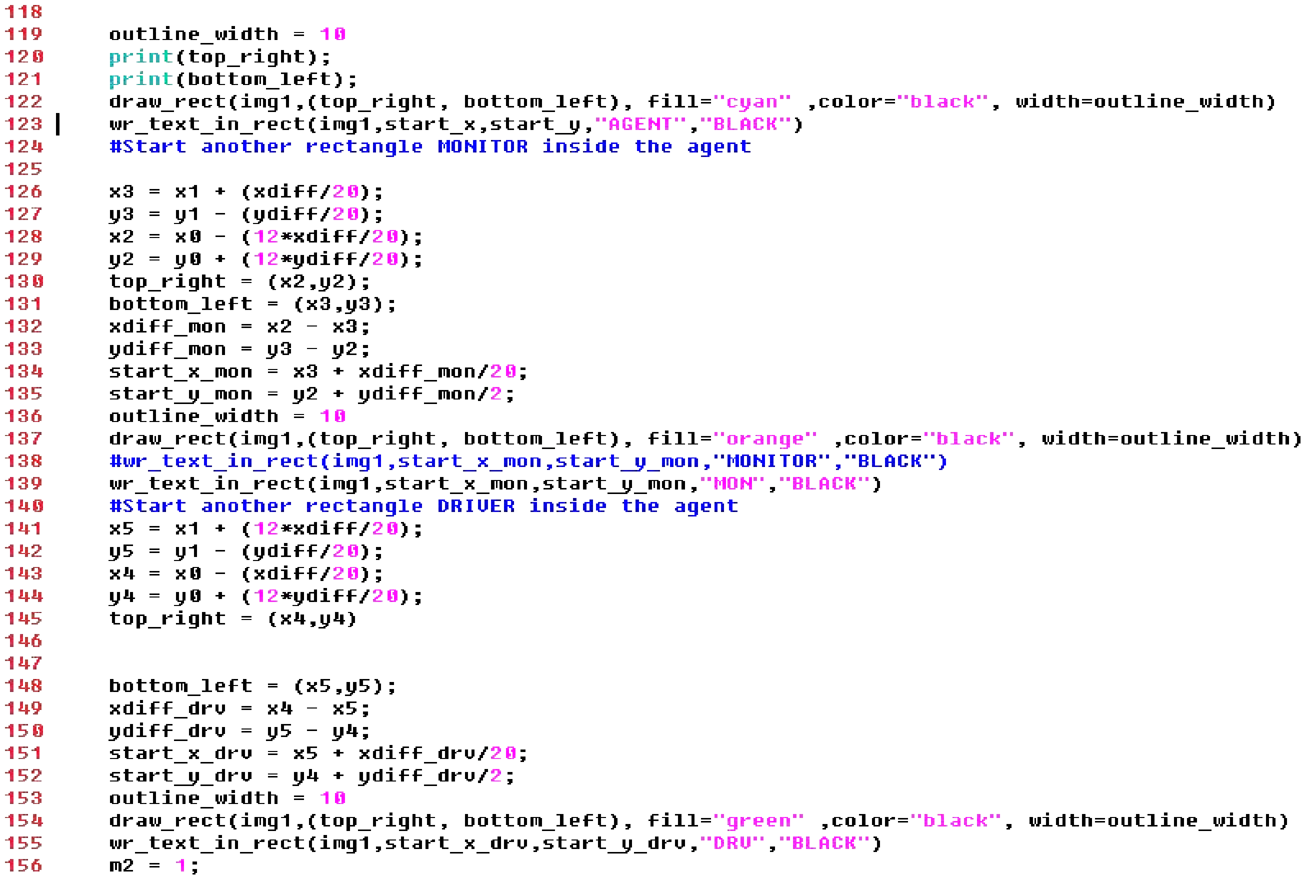
```
##########METHOD TO DRAW RECTANGLE WITH THE GIVEN CO-ORDINATES AND FILL WITH THE GIVEN COLOR##############
def draw_rect(image,coordinates,fill,color,width=1):
    rect_start = (coordinates[0][0],coordinates[0][1]);
    rect_end = (coordinates[1][0], coordinates [1][1])
    image.rectangle((rect_start,rect_end),fill=fill,outline = color)
#Method to write the text inside the rectangle
def wr_text_in_rect(image,start_wr_w,start_wr_h,str,tfill):
    font = ImageFont.truetype("arial.ttf", 15)
    image.text((start_wr_w,start_wr_h),str, fill = tfill,font = font)
#Method to draw the top,test,env blocks (w.r.t. the value of n chosen)
def call_simple_rect(w,h,n,text,bfill,tfill,img1):
    w1 = w - (n*10); #end of x should be max
    if n != 5:
        h2 = h - (n*10); #end of 'y' should be max
        h1 = n*15 + 10;
        w2 = h1;
    elif n == 5: #The height of the env block should be less; So used like below dimensions
        h2 = h - (n*10*5)
        h1 = n*15 + 10 + 35;
        w2 = n*15 + 10;
    top_right = (w1,h1)
    bottom_left = (w2,h2)
    start_x = w1 - (50);
    start_y = h1 + (n*2);
    outline_width = 10
    outline_color = "black"
    draw_rect(img1,(top_right, bottom_left), fill=bfill ,color=outline_color, width=outline_width)
    wr_text_in_rect(img1,start_x,start_y,text,tfill)
    print ("Dimensions are %0d %0d %0d %0d",top_right, bottom_left)
    return w1;
#This ■docx■ module is to manipulate with docs like MS Word. Used it to add TB diagram to the document
import docx
#This ■opencv■ module in python ease us to draw arrowed line in the image
import cv2
# This ■pillow■ module to import Image draw module
from PIL import Image,ImageDraw,ImageFont
#Taking the handle ■doc■ for docx
doc = docx.Document()
#This Module is used to measure the whole screen size
```



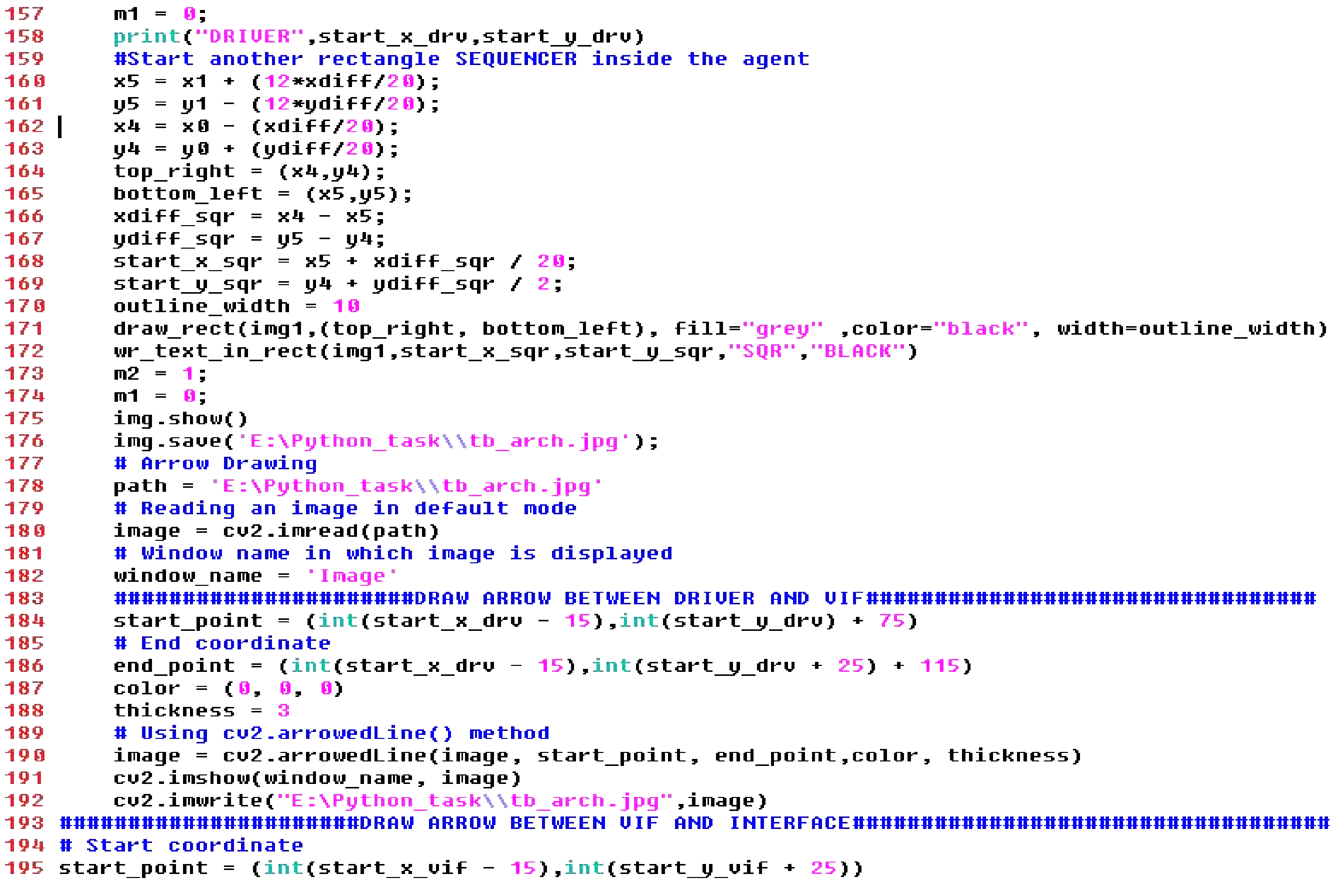
```
import tkinter
root = tkinter.Tk()
width = root.winfo_screenwidth()
height = root.winfo_screenheight()
print ("Width & HEIGHT",width,height)
# create line image of width and height
w = width
h = height
img = Image.new("RGB", (w, h),"white")
img1 = ImageDraw.Draw(img)
#Create first outer rectangle top n=1
n = 1;
top_dim = call_simple_rect(w,h,n,"TOP","orange","black",img1);
#Create second inner rectangle test n=3
n = 3;
test_dim = call_simple_rect(w,h,n,"TEST","pink","black",img1);
#Create third inner rectangle env n=5
n = 5;
env_dim = call_simple_rect(w,h,n,"ENV","yellow","black",img1);
#Create fifth inner rectangle SCOREBOARD
top_right = (w-140,150)
bottom_left = (400,230)
start_x = ((w-140)+400)/2 + 12;
start_y = 180;
draw_rect(img1,(top_right, bottom_left), fill="gray" ,color="black", width=10)
wr_text_in_rect(img1,start_x,start_y,"SCOREBOARD","BLACK")
#Create fourth inner rectangle sequences DUT
top_right = (90,h-80)
bottom_left = (w-40,h-50)
start_x_dut = (90 + (w-40))/2;
start_y_dut = (((h - 80) + (h - 50))/2) - 12;
draw_rect(img1,(top_right, bottom_left), fill="grey" ,color="black", width=10)
wr_text_in_rect(img1,start_x_dut,start_y_dut,"DUT","BLACK")
#Create fourth inner rectangle Interface
top_right = (90,h-120)
bottom_left = (w-40,h-150)
start_x_if = (90 + (w-40))/2;
start_y_if = ((h-120)+(h-150))/2 - 12;
draw_rect(img1,(top_right, bottom_left), fill="green" ,color="black", width=10)
```



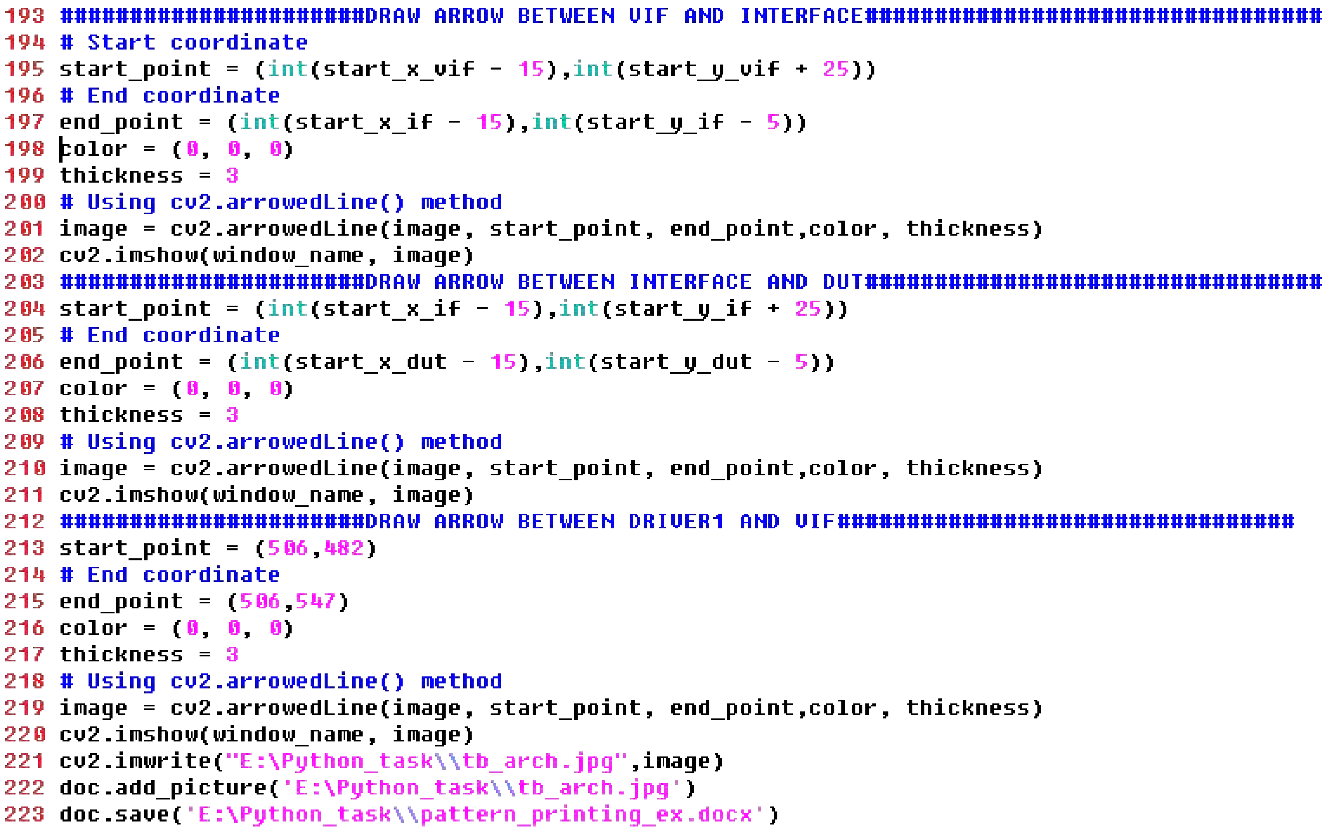
```
wr_text_in_rect(img1,start_x_if,start_y_if,"INTERFACE","BLACK")
#Create fourth inner rectangle VIF
top_right = (90,h-190)
bottom_left = (w-40,h-220)
start_x_vif = (90 + (w-40))/2;
start_y_vif = ((h-190)+(h-220))/2 - 12;
draw_rect(img1,(top_right, bottom_left), fill="gray" ,color="black", width=10)
wr_text_in_rect(img1,start_x_vif,start_y_vif,"VIF","BLACK")
print(env_dim);
#Check for number of agents
n = 7;
agnt_cnt = int(input("Enter no. of agents"))
w1 = (env_dim/agnt_cnt)
h1 = (env_dim/agnt_cnt)
tx = 40;
x0 = 0;
diff = 0;
m1 = 1;
m2 = 0;
#To draw the number of agents w.r.t. agent count
for val in range(agnt_cnt):
    print("VALUE OF X0 IS",x0)
    x1 = tx + 55;
    print("VALUE OF X1 IS",x1)
    y0 = (n*4*10)
    if agnt_cnt == 1:
        x0 = (env_dim/agnt_cnt) - 20;
    elif agnt_cnt != 1:
        x0 = (m1*(env_dim/agnt_cnt))+(m2*(x1 + diff));
    print("VALUE OF X0 LATER IS",x0)
    y1 = h - (4*n*10)
    tx = x0;
    diff = x0 - x1;
    xdiff = x0 - x1;
    ydiff = y1 - y0;
    top_right = (x0,y0);
    bottom_left = (x1,y1);
    start_x = x0 - (19*xdiff/20);
    start_y = y0 + (ydiff/20);
```



```

    outline_width = 10
    print(top_right);
    print(bottom_left);
    draw_rect(img1,(top_right, bottom_left), fill="cyan" ,color="black", width=outline_width)
    wr_text_in_rect(img1,start_x,start_y,"AGENT","BLACK")
    #Start another rectangle MONITOR inside the agent

    x3 = x1 + (xdiff/20);
    y3 = y1 - (ydiff/20);
    x2 = x0 - (12*xdiff/20);
    y2 = y0 + (12*ydiff/20);
    top_right = (x2,y2);
    bottom_left = (x3,y3);
    xdiff_mon = x2 - x3;
    ydiff_mon = y3 - y2;
    start_x_mon = x3 + xdiff_mon/20;
    start_y_mon = y2 + ydiff_mon/2;
    outline_width = 10
    draw_rect(img1,(top_right, bottom_left), fill="orange" ,color="black", width=outline_width)
    #wr_text_in_rect(img1,start_x_mon,start_y_mon,"MONITOR","BLACK")
    wr_text_in_rect(img1,start_x_mon,start_y_mon,"MON","BLACK")
    #Start another rectangle DRIVER inside the agent
    x5 = x1 + (12*xdiff/20);
    y5 = y1 - (ydiff/20);
    x4 = x0 - (xdiff/20);
    y4 = y0 + (12*ydiff/20);
    top_right = (x4,y4)


    bottom_left = (x5,y5);
    xdiff_drv = x4 - x5;
    ydiff_drv = y5 - y4;
    start_x_drv = x5 + xdiff_drv/20;
    start_y_drv = y4 + ydiff_drv/2;
    outline_width = 10
    draw_rect(img1,(top_right, bottom_left), fill="green" ,color="black", width=outline_width)
    wr_text_in_rect(img1,start_x_drv,start_y_drv,"DRV","BLACK")
    m2 = 1;
```

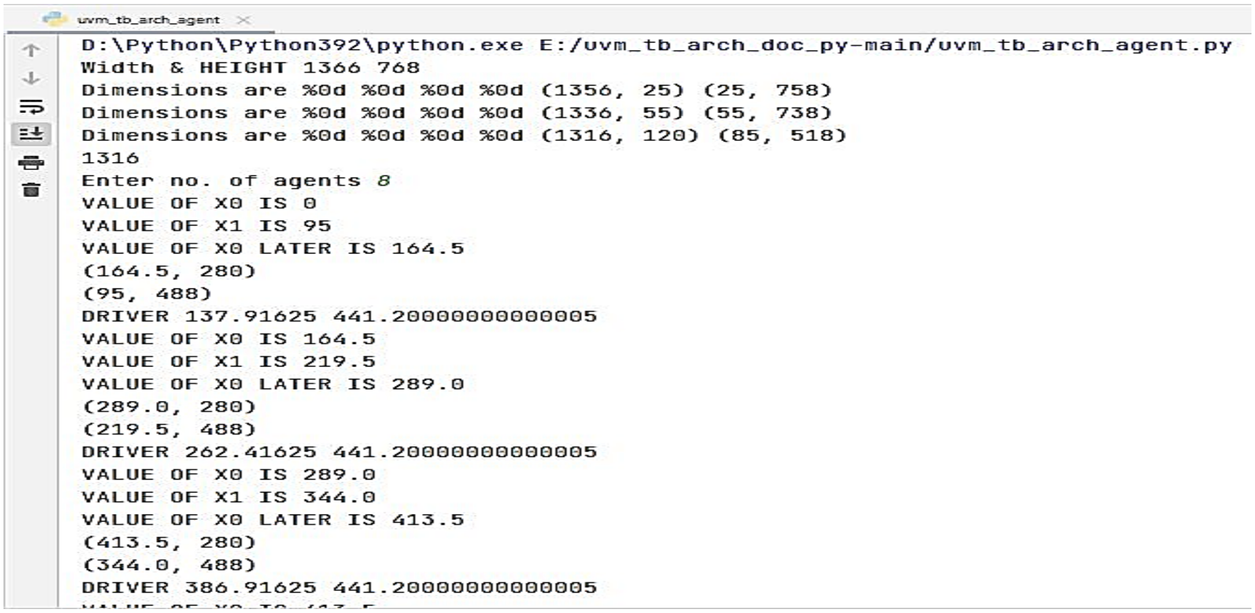


```
    m1 = 0;
    print("DRIVER",start_x_drv,start_y_drv)
    #Start another rectangle SEQUENCER inside the agent
    x5 = x1 + (12*xdiff/20);
    y5 = y1 - (12*ydiff/20);
    x4 = x0 - (xdiff/20);
    y4 = y0 + (ydiff/20);
    top_right = (x4,y4);
    bottom_left = (x5,y5);
    xdiff_sqr = x4 - x5;
    ydiff_sqr = y5 - y4;
    start_x_sqr = x5 + xdiff_sqr / 20;
    start_y_sqr = y4 + ydiff_sqr / 2;
    outline_width = 10
    draw_rect(img1,(top_right, bottom_left), fill="grey" ,color="black", width=outline_width)
    wr_text_in_rect(img1,start_x_sqr,start_y_sqr,"SQR","BLACK")
    m2 = 1;
    m1 = 0;
    img.show()
    img.save('E:\Python_task\\tb_arch.jpg');
    # Arrow Drawing
    path = 'E:\Python_task\\tb_arch.jpg'
    # Reading an image in default mode
    image = cv2.imread(path)
    # Window name in which image is displayed
    window_name = 'Image'
    ######################DRAW ARROW BETWEEN DRIVER AND VIF#################################
    start_point = (int(start_x_drv - 15),int(start_y_drv) + 75)
    # End coordinate
    end_point = (int(start_x_drv - 15),int(start_y_drv + 25) + 115)
    color = (0, 0, 0)
    thickness = 3
    # Using cv2.arrowedLine() method
    image = cv2.arrowedLine(image, start_point, end_point,color, thickness)
    cv2.imshow(window_name, image)
    cv2.imwrite("E:\Python_task\\tb_arch.jpg",image)
######################DRAW ARROW BETWEEN VIF AND INTERFACE#################################
# Start coordinate
start_point = (int(start_x_vif - 15),int(start_y_vif + 25))
```

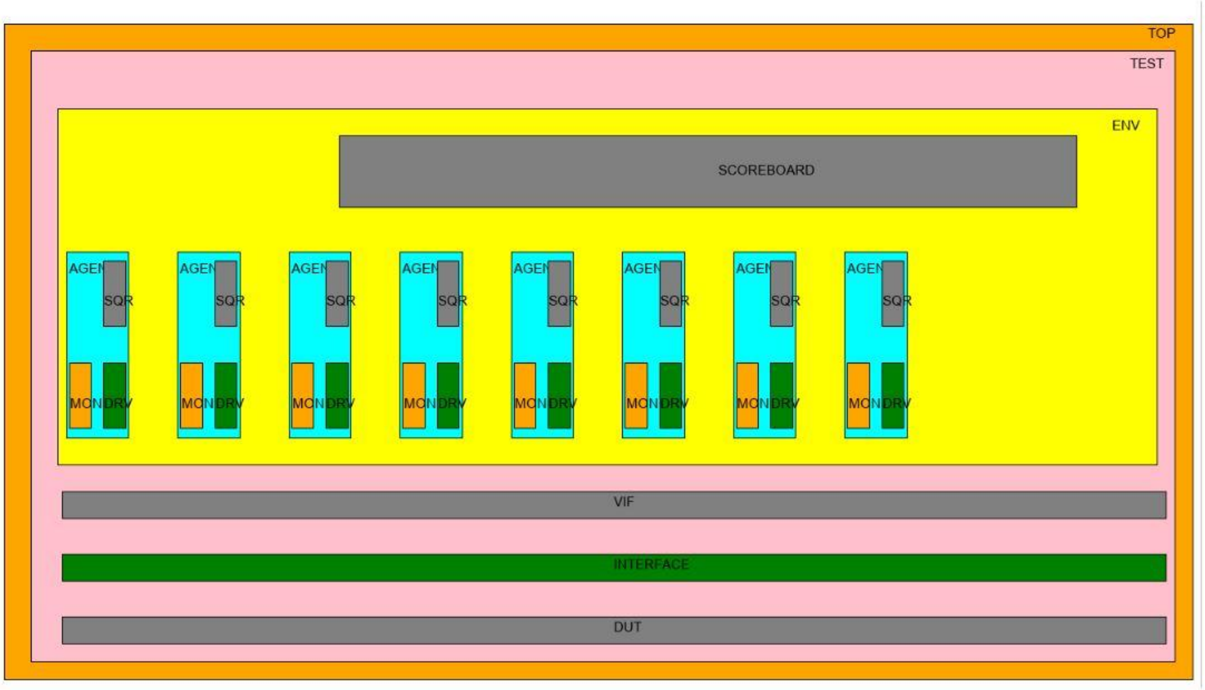


```
######################DRAW ARROW BETWEEN VIF AND INTERFACE#################################
# Start coordinate
start_point = (int(start_x_vif - 15),int(start_y_vif + 25))
# End coordinate
end_point = (int(start_x_if - 15),int(start_y_if - 5))
color = (0, 0, 0)
thickness = 3
# Using cv2.arrowedLine() method
image = cv2.arrowedLine(image, start_point, end_point,color, thickness)
cv2.imshow(window_name, image)
######################DRAW ARROW BETWEEN INTERFACE AND DUT#################################
start_point = (int(start_x_if - 15),int(start_y_if + 25))
# End coordinate
end_point = (int(start_x_dut - 15),int(start_y_dut - 5))
color = (0, 0, 0)
thickness = 3
# Using cv2.arrowedLine() method
image = cv2.arrowedLine(image, start_point, end_point,color, thickness)
cv2.imshow(window_name, image)
######################DRAW ARROW BETWEEN DRIVER1 AND VIF#################################
start_point = (506,482)
# End coordinate
end_point = (506,547)
color = (0, 0, 0)
thickness = 3
# Using cv2.arrowedLine() method
image = cv2.arrowedLine(image, start_point, end_point,color, thickness)
cv2.imshow(window_name, image)
cv2.imwrite("E:\Python_task\\tb_arch.jpg",image)
doc.add_picture('E:\Python_task\\tb_arch.jpg')
doc.save('E:\Python_task\\pattern_printing_ex.docx')
```

Results of uvm_tb_arch_agent.py

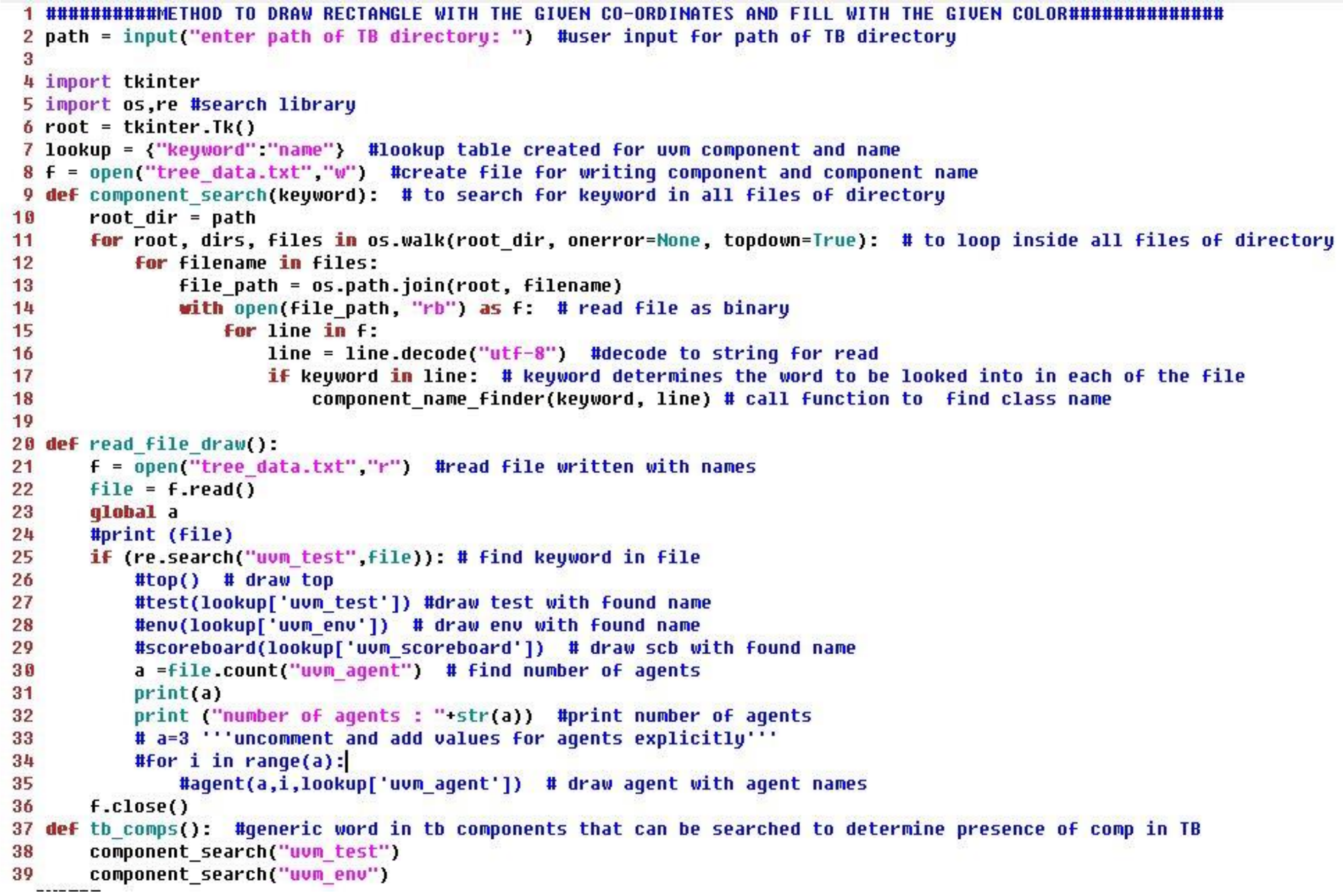


```
D:\Python\Python392\python.exe E:/uvm_tb_arch_doc_py-main/uvm_tb_arch_agent.py
Width & HEIGHT 1366 768
Dimensions are %0d %0d %0d %0d (1356, 25) (25, 758)
Dimensions are %0d %0d %0d %0d (1336, 55) (55, 738)
Dimensions are %0d %0d %0d %0d (1316, 120) (85, 518)
1316
Enter no. of agents 8
VALUE OF X0 IS 0
VALUE OF X1 IS 95
VALUE OF X0 LATER IS 164.5
(164.5, 280)
(95, 488)
DRIVER 137.91625 441.20000000000005
VALUE OF X0 IS 164.5
VALUE OF X1 IS 219.5
VALUE OF X0 LATER IS 289.0
(289.0, 280)
(219.5, 488)
DRIVER 262.41625 441.20000000000005
VALUE OF X0 IS 289.0
VALUE OF X1 IS 344.0
VALUE OF X0 LATER IS 413.5
(413.5, 280)
(344.0, 488)
DRIVER 386.91625 441.20000000000005
VALUE OF X0 IS 413.5
```

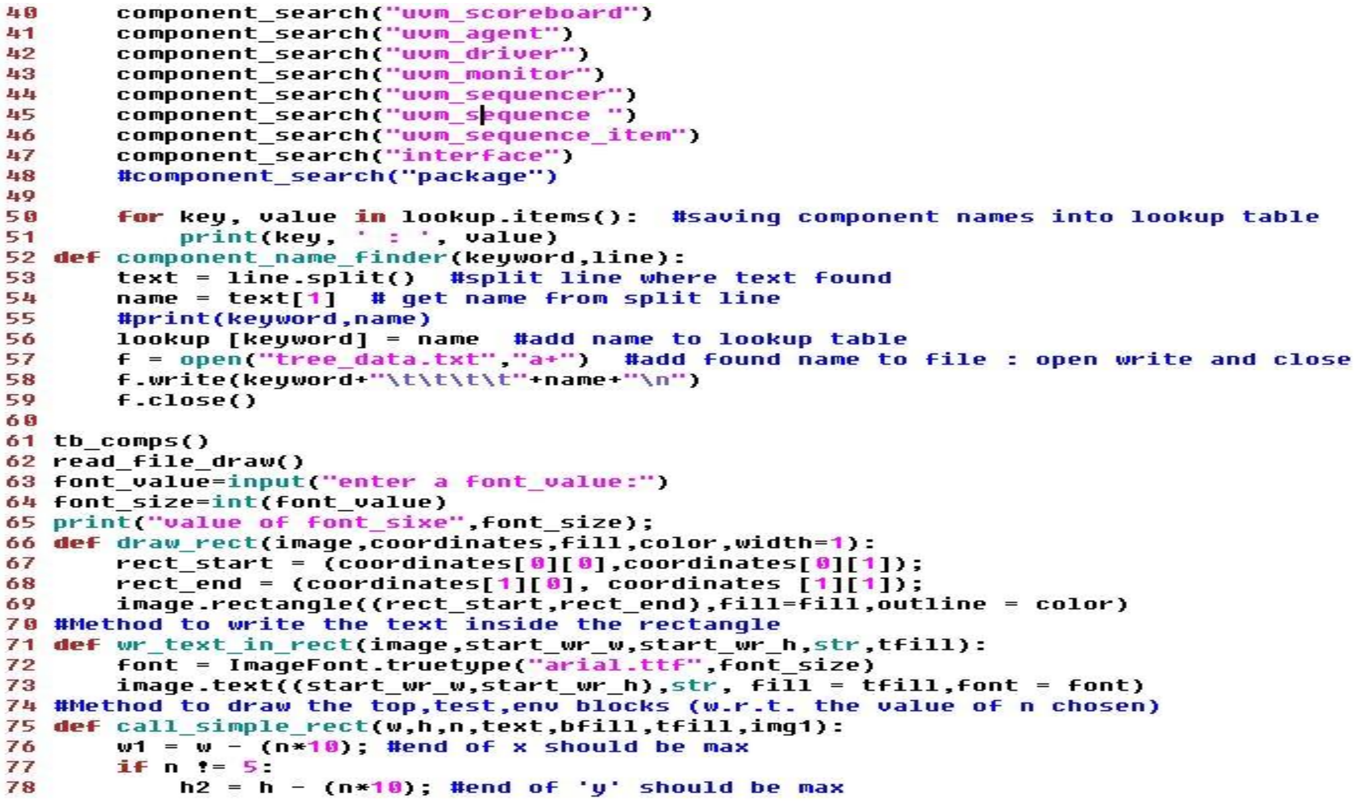


3.8 uvm_tb_name_image_file_generator.py

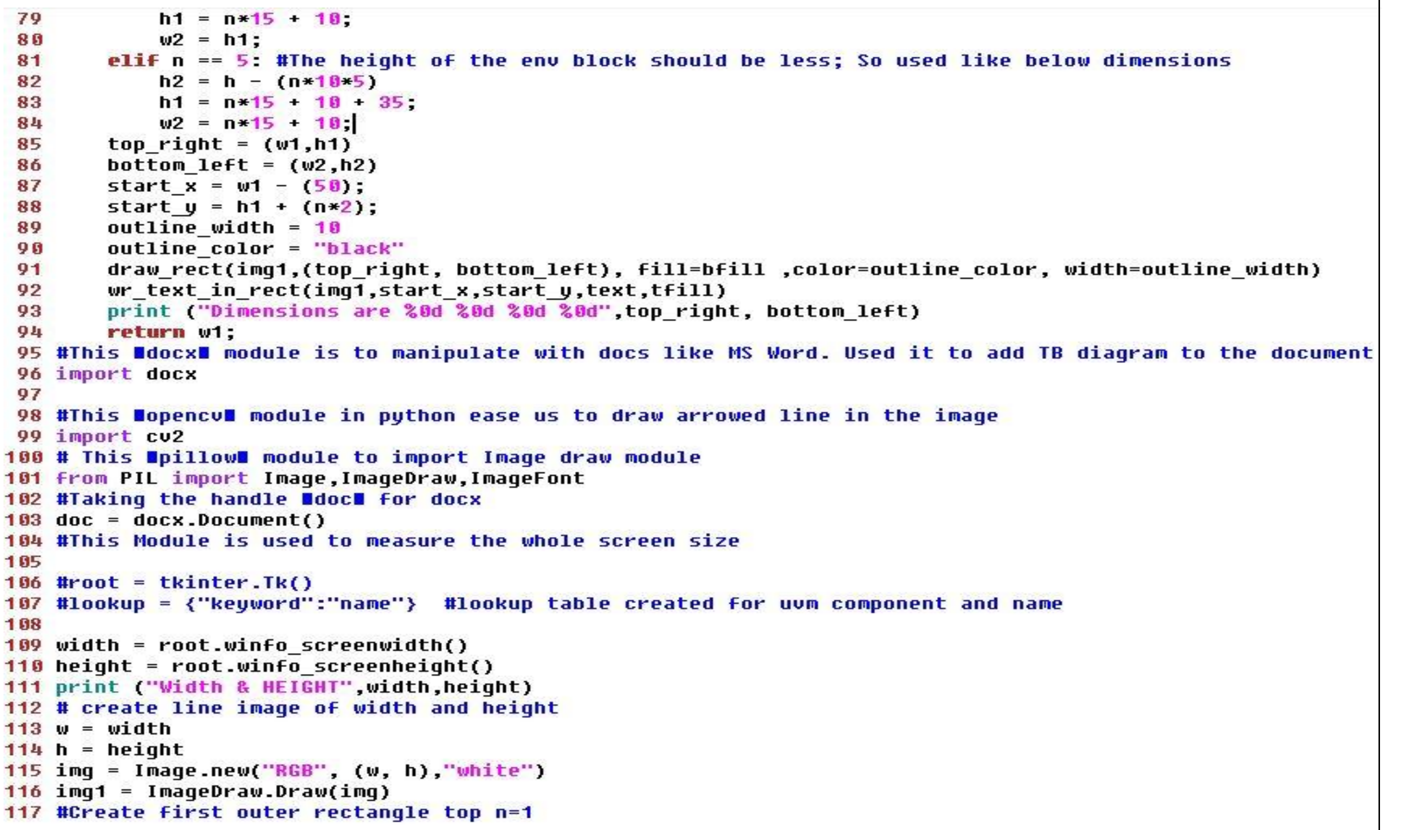
- Firstly import the PIL(Python Imaging Library) which is Pillow, it adds image processing capabilities to python interpreter.
- This library provides extensive file format support, an efficient internal representation, and fairly powerful image processing capabilities. Import tkinter package (tk interface) is a standard python interface to the tk GUI toolkit.
- Now adding pillow and tkinter for image draw and canvas. The Canvas is a rectangular area intended for drawing pictures or other complex layouts. Import image , imagedraw and imagefont from pillow.
- Then import OS library which is the module in python provides functions for interacting with the operating system.this module provides a portable way of using operating system-dependent functionality.
- Now user enters the path of TB directory. Set width and height according to the screen size. Set canvas size, create lookup table for uvm component and name. Create file for writing component and component name, also clean file before writing the data.
- Now drawing top level structure, set the co-ordinates and set color for top as yellow color. Drawing test level structure, set the co-ordinates and set color for top as orange color. Drawing env level structure, set the co-ordinates and set color for top as yellow color.
- Drawing scoreboard level structure, set the co-ordinates and set color for top as red. Drawing agent level structure, set the co-ordinates and set color for top as blue color also set the image font.
- Call the rectangular creation function. So we are doing here, user enter the input as TB directory and it will create block diagram of the component like- top, test, env, agent, scoreboard



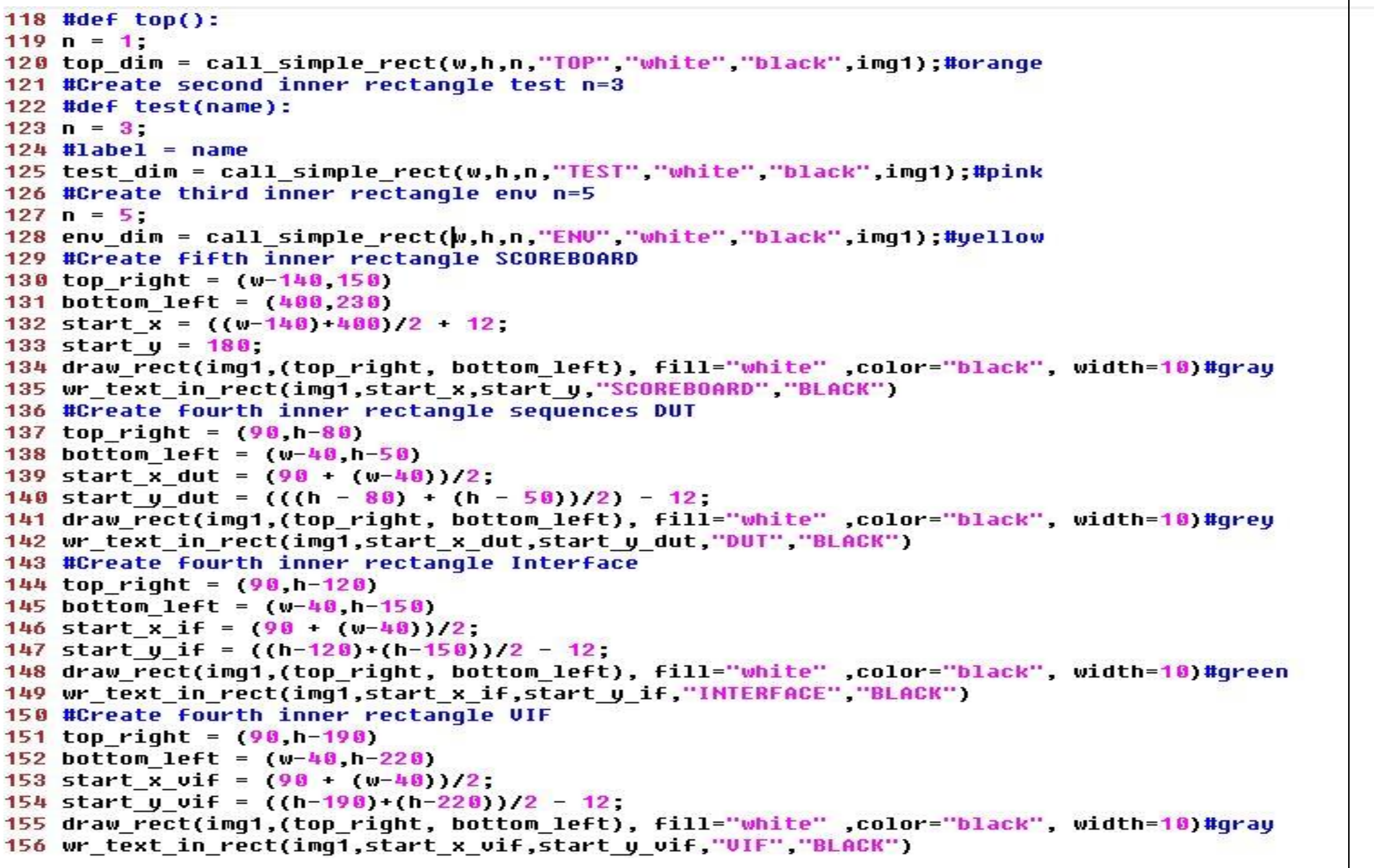
```
##########METHOD TO DRAW RECTANGLE WITH THE GIVEN CO-ORDINATES AND FILL WITH THE GIVEN COLOR#############
path = input("enter path of TB directory: ")  #user input for path of TB directory

import tkinter
import os,re #search library
root = tkinter.Tk()
lookup = {"keyword":"name"}  #lookup table created for uvm component and name
f = open("tree_data.txt","w")  #create file for writing component and component name
def component_search(keyword):  # to search for keyword in all files of directory
    root_dir = path
    for root, dirs, files in os.walk(root_dir, onerror=None, topdown=True):  # to loop inside all files of directory
        for filename in files:
            file_path = os.path.join(root, filename)
            with open(file_path, "rb") as f:  # read file as binary
                for line in f:
                    line = line.decode("utf-8")  #decode to string for read
                    if keyword in line:  # keyword determines the word to be looked into in each of the file
                        component_name_finder(keyword, line) # call function to  find class name

def read_file_draw():
    f = open("tree_data.txt","r")  #read file written with names
    file = f.read()
    global a
    #print (file)
    if (re.search("uvm_test",file)): # find keyword in file
        #top()  # draw top
        #test(lookup['uvm_test']) #draw test with found name
        #env(lookup['uvm_env'])  # draw env with found name
        #scoreboard(lookup['uvm_scoreboard'])  # draw scb with found name
        a =file.count("uvm_agent")  # find number of agents
        print(a)
        print ("number of agents : "+str(a))  #print number of agents
        # a=3 '''uncomment and add values for agents explicitly'''
        #for i in range(a):
            #agent(a,i,lookup['uvm_agent'])  # draw agent with agent names
    f.close()
def tb_comps():  #generic word in tb components that can be searched to determine presence of comp in TB
    component_search("uvm_test")
    component_search("uvm_env")
```



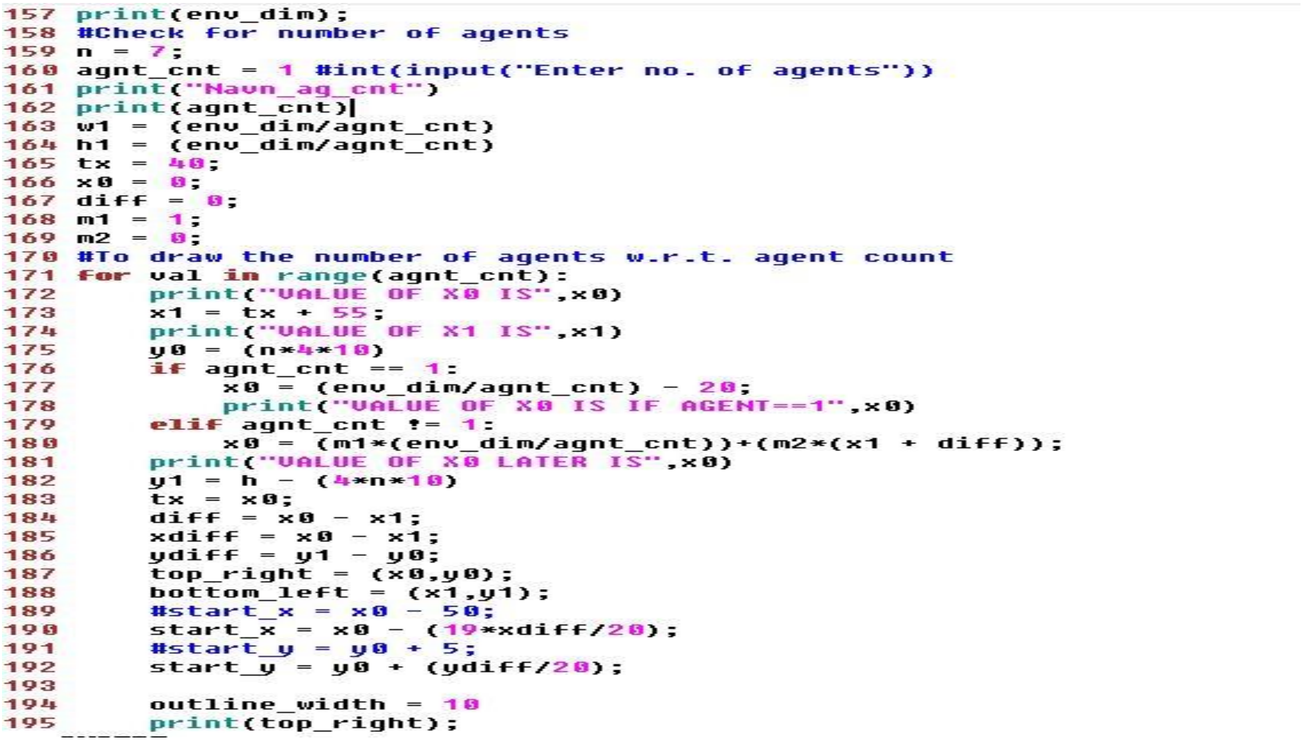
```
    component_search("uvm_scoreboard")
    component_search("uvm_agent")
    component_search("uvm_driver")
    component_search("uvm_monitor")
    component_search("uvm_sequencer")
    component_search("uvm_sequence ")
    component_search("uvm_sequence_item")
    component_search("interface")
    #component_search("package")

    for key, value in lookup.items():  #saving component names into lookup table
        print(key, ' : ', value)
def component_name_finder(keyword,line):
    text = line.split()  #split line where text found
    name = text[1]  # get name from split line
    #print(keyword,name)
    lookup [keyword] = name  #add name to lookup table
    f = open("tree_data.txt","a+")  #add found name to file : open write and close
    f.write(keyword+"\t\t\t\t"+name+"\n")
    f.close()

tb_comps()
read_file_draw()
font_value=input("enter a font_value:")
font_size=int(font_value)
print("value of font_sixe",font_size);
def draw_rect(image,coordinates,fill,color,width=1):
    rect_start = (coordinates[0][0],coordinates[0][1]);
    rect_end = (coordinates[1][0], coordinates [1][1]);
    image.rectangle((rect_start,rect_end),fill=fill,outline = color)
#Method to write the text inside the rectangle
def wr_text_in_rect(image,start_wr_w,start_wr_h,str,tfill):
    font = ImageFont.truetype("arial.ttf",font_size)
    image.text((start_wr_w,start_wr_h),str, fill = tfill,font = font)
#Method to draw the top,test,env blocks (w.r.t. the value of n chosen)
def call_simple_rect(w,h,n,text,bfill,tfill,img1):
    w1 = w - (n*10); #end of x should be max
    if n != 5:
        h2 = h - (n*10); #end of 'y' should be max
```



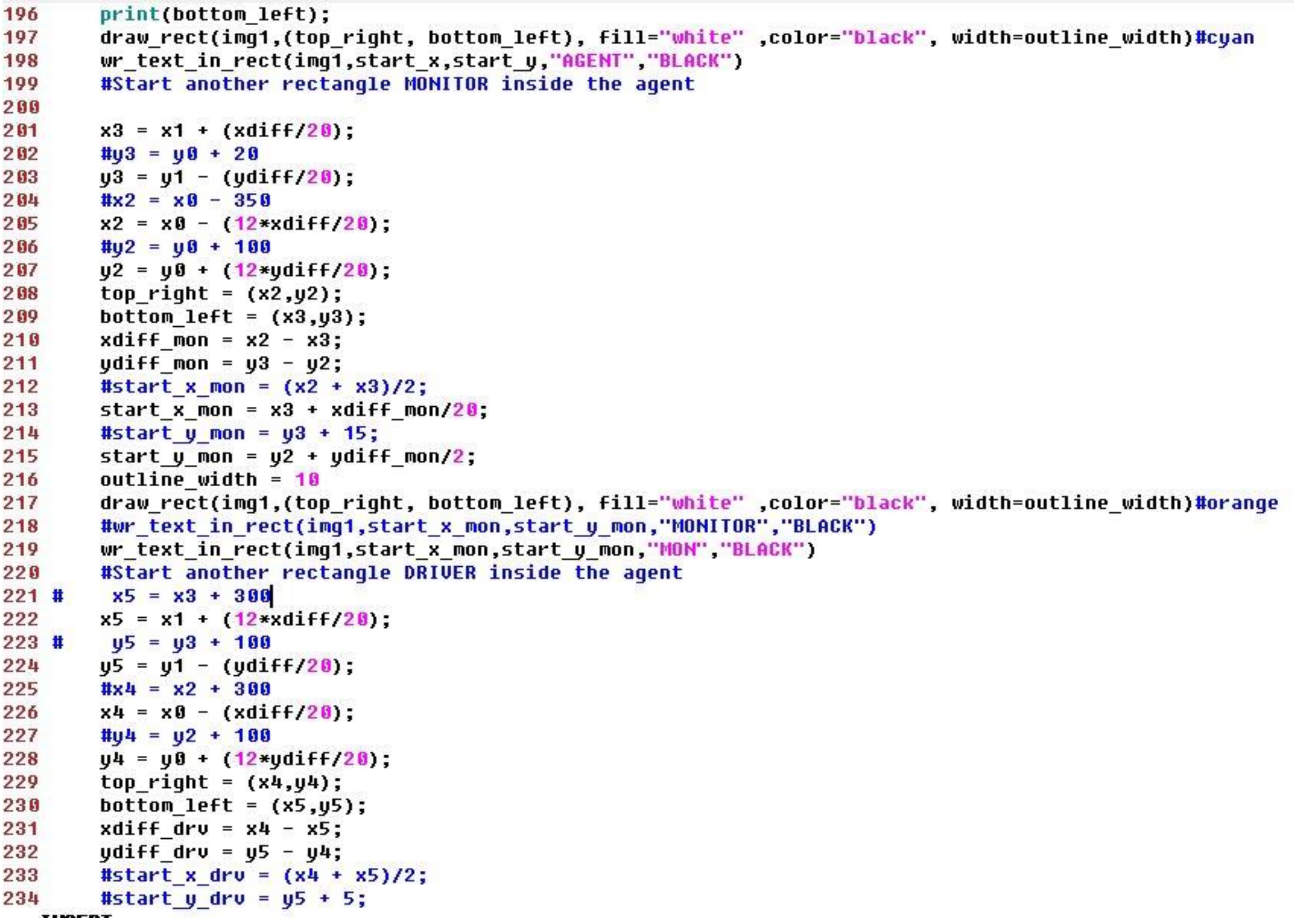
```
        h1 = n*15 + 10;
        w2 = h1;
    elif n == 5: #The height of the env block should be less; So used like below dimensions
        h2 = h - (n*10*5)
        h1 = n*15 + 10 + 35;
        w2 = n*15 + 10;
    top_right = (w1,h1)
    bottom_left = (w2,h2)
    start_x = w1 - (50);
    start_y = h1 + (n*2);
    outline_width = 10
    outline_color = "black"
    draw_rect(img1,(top_right, bottom_left), fill=bfill ,color=outline_color, width=outline_width)
    wr_text_in_rect(img1,start_x,start_y,text,tfill)
    print ("Dimensions are %0d %0d %0d %0d",top_right, bottom_left)
    return w1;
#This ■docx■ module is to manipulate with docs like MS Word. Used it to add TB diagram to the document
import docx

#This ■opencv■ module in python ease us to draw arrowed line in the image
import cv2
# This ■pillow■ module to import Image draw module
from PIL import Image,ImageDraw,ImageFont
#Taking the handle ■doc■ for docx
doc = docx.Document()
#This Module is used to measure the whole screen size

#root = tkinter.Tk()
#lookup = {"keyword":"name"}  #lookup table created for uvm component and name

width = root.winfo_screenwidth()
height = root.winfo_screenheight()
print ("Width & HEIGHT",width,height)
# create line image of width and height
w = width
h = height
img = Image.new("RGB", (w, h),"white")
img1 = ImageDraw.Draw(img)
#Create first outer rectangle top n=1
```



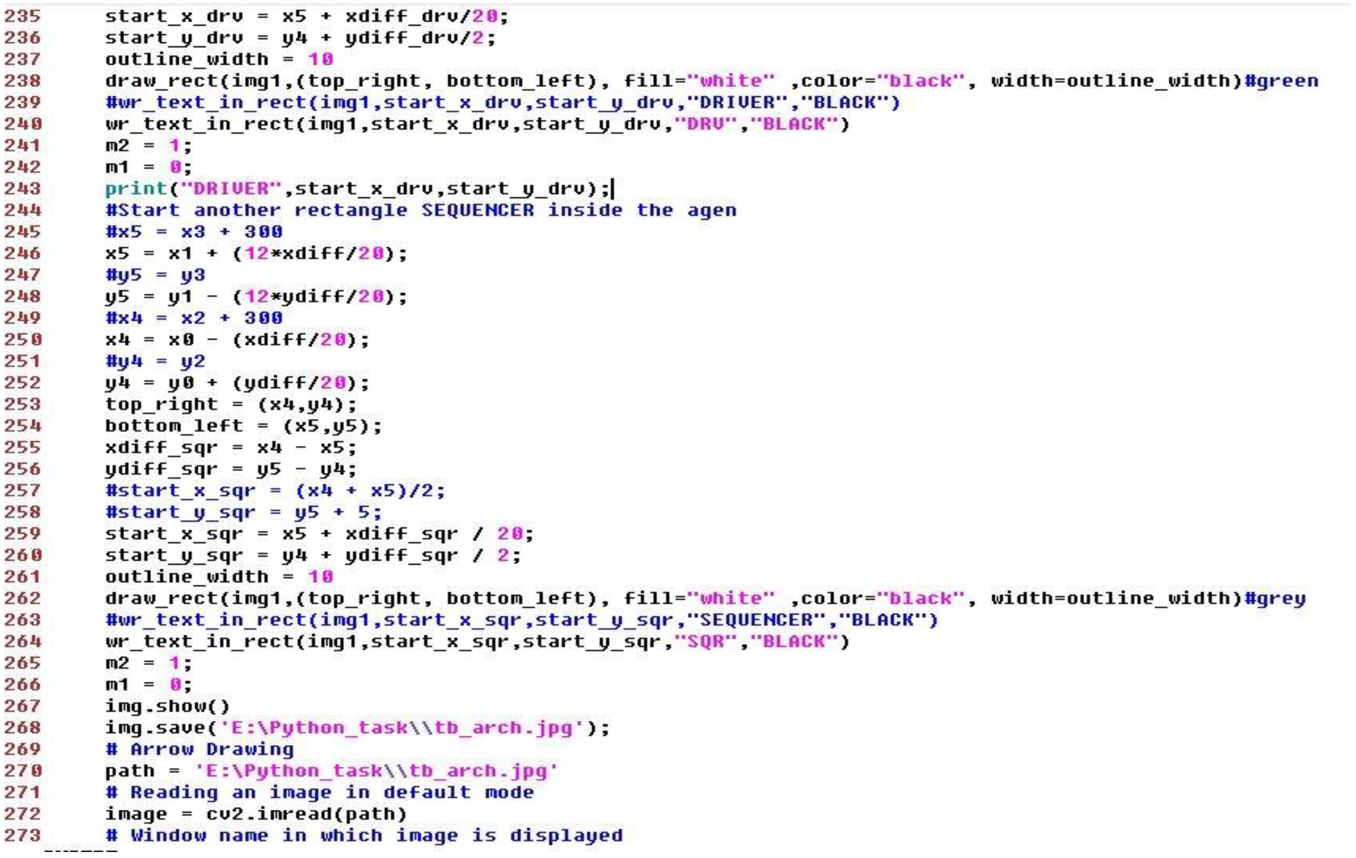
```
#def top():
n = 1;
top_dim = call_simple_rect(w,h,n,"TOP","white","black",img1);#orange
#Create second inner rectangle test n=3
#def test(name):
n = 3;
#label = name
test_dim = call_simple_rect(w,h,n,"TEST","white","black",img1);#pink
#Create third inner rectangle env n=5
n = 5;
env_dim = call_simple_rect(w,h,n,"ENV","white","black",img1);#yellow
#Create fifth inner rectangle SCOREBOARD
top_right = (w-140,150)
bottom_left = (400,230)
start_x = ((w-140)+400)/2 + 12;
start_y = 180;
draw_rect(img1,(top_right, bottom_left), fill="white" ,color="black", width=10)#gray
wr_text_in_rect(img1,start_x,start_y,"SCOREBOARD","BLACK")
#Create fourth inner rectangle sequences DUT
top_right = (90,h-80)
bottom_left = (w-40,h-50)
start_x_dut = (90 + (w-40))/2;
start_y_dut = (((h - 80) + (h - 50))/2) - 12;
draw_rect(img1,(top_right, bottom_left), fill="white" ,color="black", width=10)#grey
wr_text_in_rect(img1,start_x_dut,start_y_dut,"DUT","BLACK")
#Create fourth inner rectangle Interface
top_right = (90,h-120)
bottom_left = (w-40,h-150)
start_x_if = (90 + (w-40))/2;
start_y_if = ((h-120)+(h-150))/2 - 12;
draw_rect(img1,(top_right, bottom_left), fill="white" ,color="black", width=10)#green
wr_text_in_rect(img1,start_x_if,start_y_if,"INTERFACE","BLACK")
#Create fourth inner rectangle VIF
top_right = (90,h-190)
bottom_left = (w-40,h-220)
start_x_vif = (90 + (w-40))/2;
start_y_vif = ((h-190)+(h-220))/2 - 12;
draw_rect(img1,(top_right, bottom_left), fill="white" ,color="black", width=10)#gray
wr_text_in_rect(img1,start_x_vif,start_y_vif,"VIF","BLACK")
```



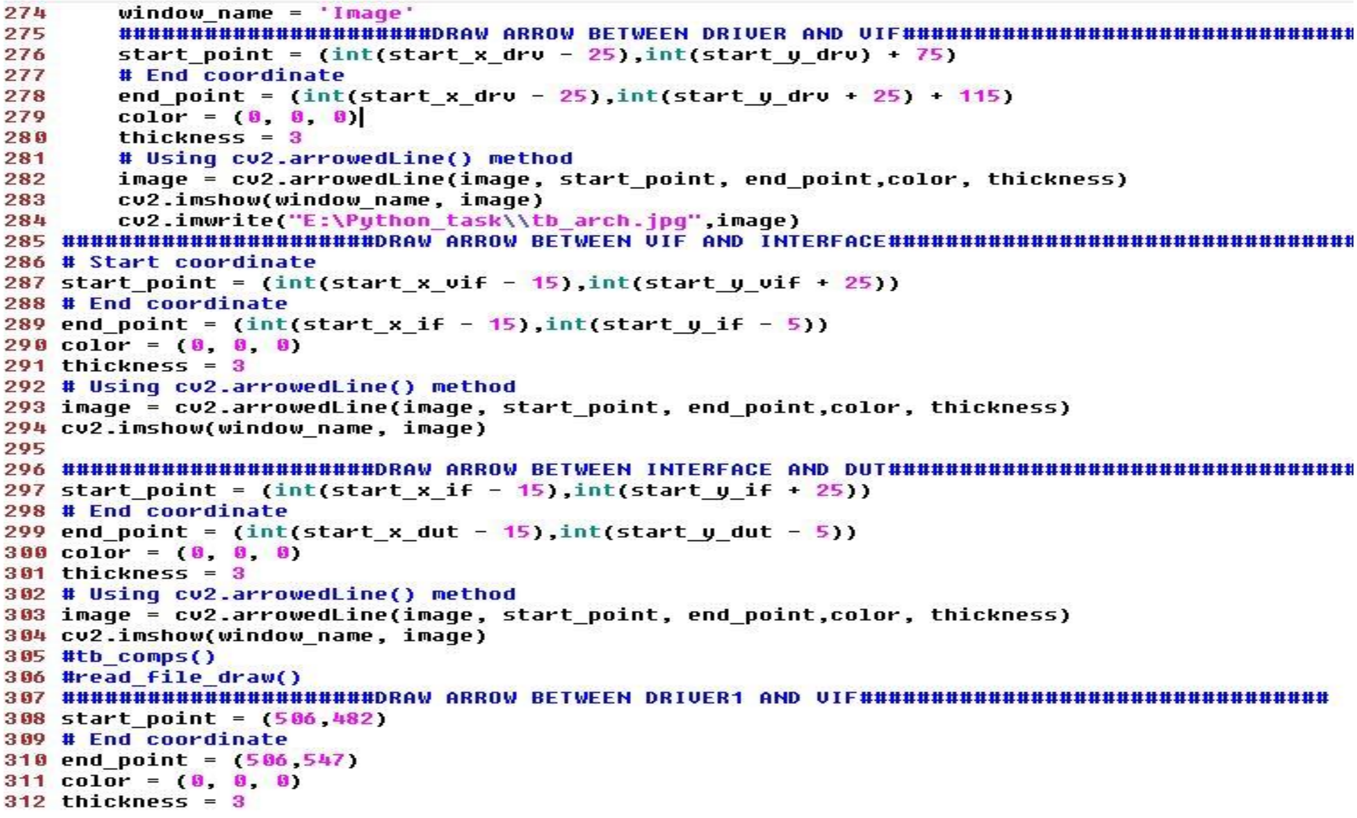
```
print(env_dim);
#Check for number of agents
n = 7;
agnt_cnt = 1 #int(input("Enter no. of agents"))
print("Navn_ag_cnt")
print(agnt_cnt)
w1 = (env_dim/agnt_cnt)
h1 = (env_dim/agnt_cnt)
tx = 40;
x0 = 0;
diff = 0;
m1 = 1;
m2 = 0;
#To draw the number of agents w.r.t. agent count
for val in range(agnt_cnt):
    print("VALUE OF X0 IS",x0)
    x1 = tx + 55;
    print("VALUE OF X1 IS",x1)
    y0 = (n*4*10)
    if agnt_cnt == 1:
        x0 = (env_dim/agnt_cnt) - 20;
        print("VALUE OF X0 IS IF AGENT==1",x0)
    elif agnt_cnt != 1:
        x0 = (m1*(env_dim/agnt_cnt))+(m2*(x1 + diff));
    print("VALUE OF X0 LATER IS",x0)
    y1 = h - (4*n*10)
    tx = x0;
    diff = x0 - x1;
    xdiff = x0 - x1;
    ydiff = y1 - y0;
    top_right = (x0,y0);
    bottom_left = (x1,y1);
    #start_x = x0 - 50;
    start_x = x0 - (19*xdiff/20);
    #start_y = y0 + 5;
    start_y = y0 + (ydiff/20);

    outline_width = 10
    print(top_right);
```



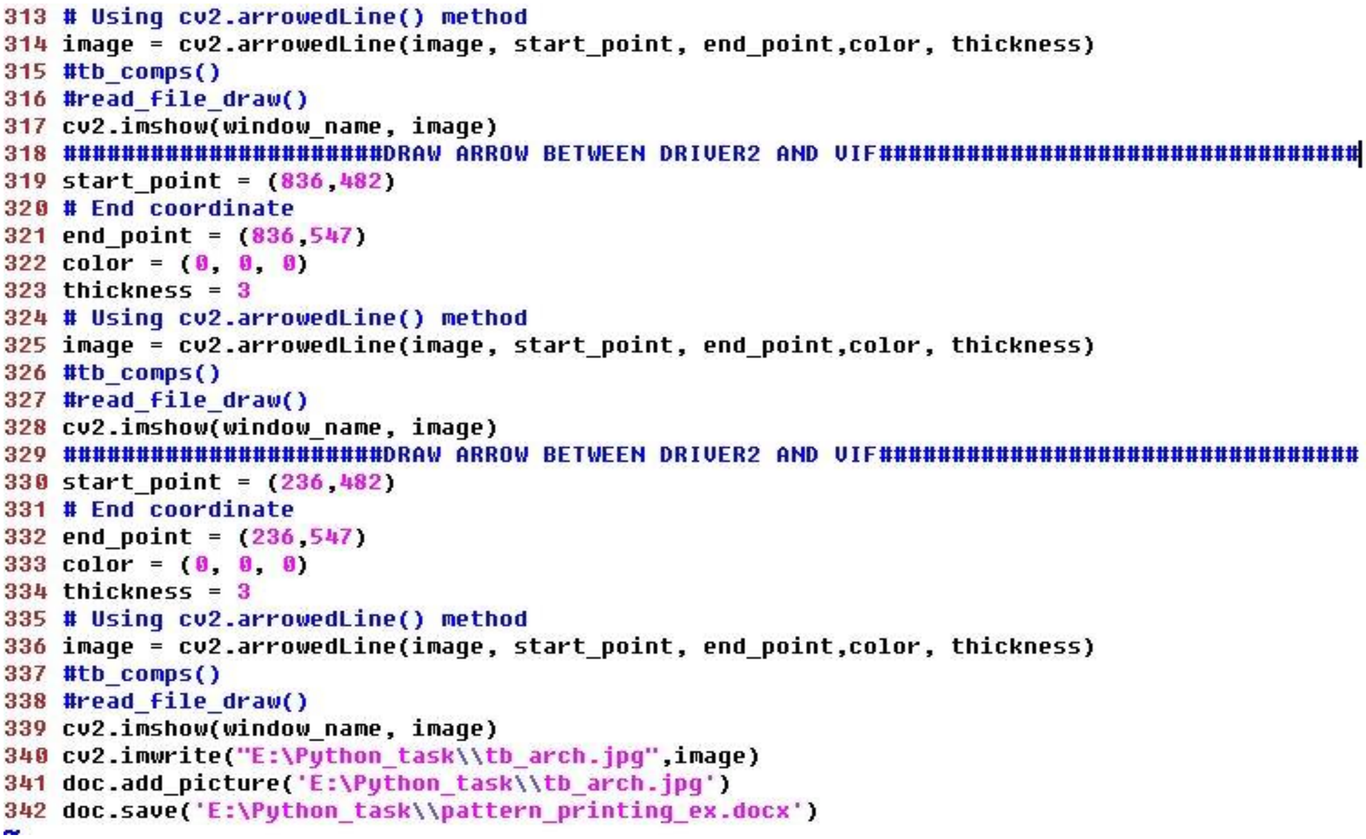
```
    print(bottom_left);
    draw_rect(img1,(top_right, bottom_left), fill="white" ,color="black", width=outline_width)#cyan
    wr_text_in_rect(img1,start_x,start_y,"AGENT","BLACK")
    #Start another rectangle MONITOR inside the agent

    x3 = x1 + (xdiff/20);
    #y3 = y0 + 20
    y3 = y1 - (ydiff/20);
    #x2 = x0 - 350
    x2 = x0 - (12*xdiff/20);
    #y2 = y0 + 100
    y2 = y0 + (12*ydiff/20);
    top_right = (x2,y2);
    bottom_left = (x3,y3);
    xdiff_mon = x2 - x3;
    ydiff_mon = y3 - y2;
    #start_x_mon = (x2 + x3)/2;
    start_x_mon = x3 + xdiff_mon/20;
    #start_y_mon = y3 + 15;
    start_y_mon = y2 + ydiff_mon/2;
    outline_width = 10
    draw_rect(img1,(top_right, bottom_left), fill="white" ,color="black", width=outline_width)#orange
    #wr_text_in_rect(img1,start_x_mon,start_y_mon,"MONITOR","BLACK")
    wr_text_in_rect(img1,start_x_mon,start_y_mon,"MON","BLACK")
    #Start another rectangle DRIVER inside the agent
#    x5 = x3 + 300
    x5 = x1 + (12*xdiff/20);
#    y5 = y3 + 100
    y5 = y1 - (ydiff/20);
    #x4 = x2 + 300
    x4 = x0 - (xdiff/20);
    #y4 = y2 + 100
    y4 = y0 + (12*ydiff/20);
    top_right = (x4,y4);
    bottom_left = (x5,y5);
    xdiff_drv = x4 - x5;
    ydiff_drv = y5 - y4;
    #start_x_drv = (x4 + x5)/2;
    #start_y_drv = y5 + 5;
```



```
    start_x_drv = x5 + xdiff_drv/20;
    start_y_drv = y4 + ydiff_drv/2;
    outline_width = 10
    draw_rect(img1,(top_right, bottom_left), fill="white" ,color="black", width=outline_width)#green
    #wr_text_in_rect(img1,start_x_drv,start_y_drv,"DRIVER","BLACK")
    wr_text_in_rect(img1,start_x_drv,start_y_drv,"DRV","BLACK")
    m2 = 1;
    m1 = 0;
    print("DRIVER",start_x_drv,start_y_drv);
    #Start another rectangle SEQUENCER inside the agen
    #x5 = x3 + 300
    x5 = x1 + (12*xdiff/20);
    #y5 = y3
    y5 = y1 - (12*ydiff/20);
    #x4 = x2 + 300
    x4 = x0 - (xdiff/20);
    #y4 = y2
    y4 = y0 + (ydiff/20);
    top_right = (x4,y4);
    bottom_left = (x5,y5);
    xdiff_sqr = x4 - x5;
    ydiff_sqr = y5 - y4;
    #start_x_sqr = (x4 + x5)/2;
    #start_y_sqr = y5 + 5;
    start_x_sqr = x5 + xdiff_sqr / 20;
    start_y_sqr = y4 + ydiff_sqr / 2;
    outline_width = 10
    draw_rect(img1,(top_right, bottom_left), fill="white" ,color="black", width=outline_width)#grey
    #wr_text_in_rect(img1,start_x_sqr,start_y_sqr,"SEQUENCER","BLACK")
    wr_text_in_rect(img1,start_x_sqr,start_y_sqr,"SQR","BLACK")
    m2 = 1;
    m1 = 0;
    img.show()
    img.save('E:\Python_task\\tb_arch.jpg');
    # Arrow Drawing
    path = 'E:\Python_task\\tb_arch.jpg'
    # Reading an image in default mode
    image = cv2.imread(path)
    # Window name in which image is displayed
```

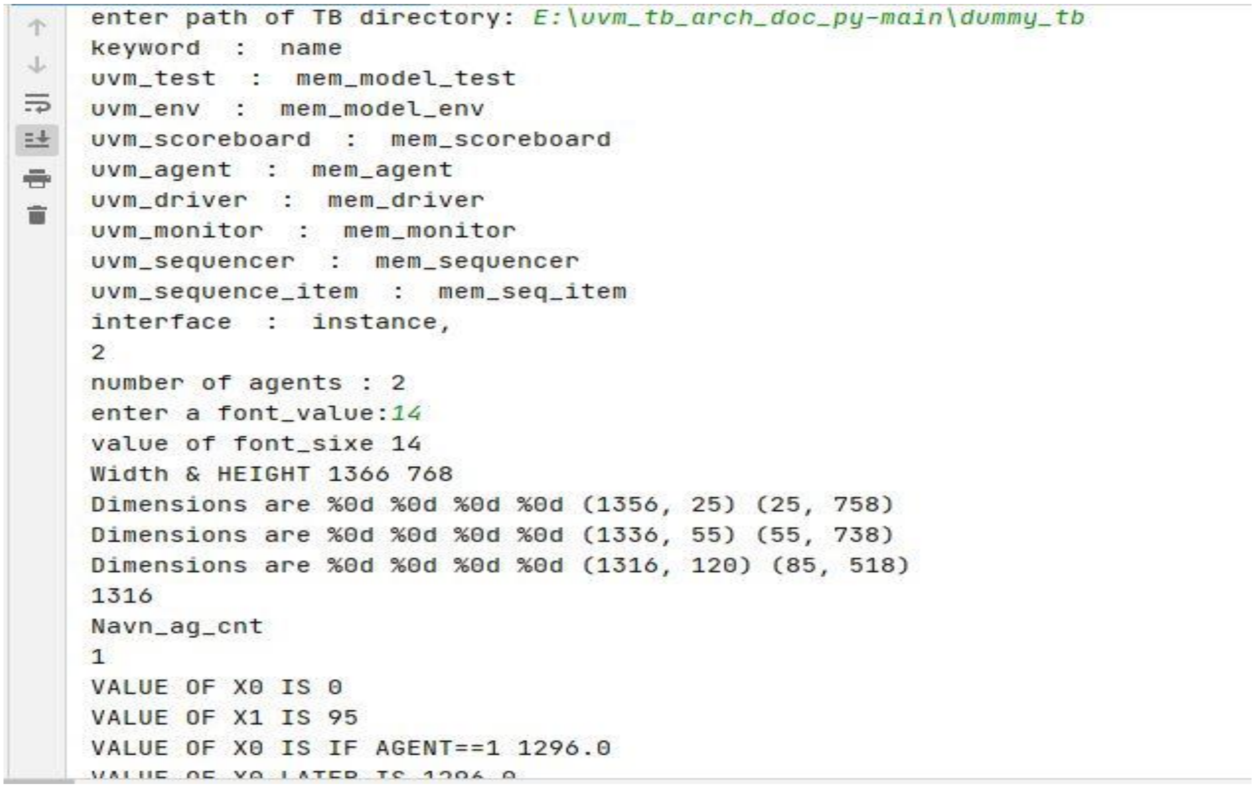


```
    window_name = 'Image'
    ######################DRAW ARROW BETWEEN DRIVER AND VIF################################
    start_point = (int(start_x_drv - 25),int(start_y_drv) + 75)
    # End coordinate
    end_point = (int(start_x_drv - 25),int(start_y_drv + 25) + 115)
    color = (0, 0, 0)
    thickness = 3
    # Using cv2.arrowedLine() method
    image = cv2.arrowedLine(image, start_point, end_point,color, thickness)
    cv2.imshow(window_name, image)
    cv2.imwrite("E:\Python_task\\tb_arch.jpg",image)
######################DRAW ARROW BETWEEN VIF AND INTERFACE#################################
# Start coordinate
start_point = (int(start_x_vif - 15),int(start_y_vif + 25))
# End coordinate
end_point = (int(start_x_if - 15),int(start_y_if - 5))
color = (0, 0, 0)
thickness = 3
# Using cv2.arrowedLine() method
image = cv2.arrowedLine(image, start_point, end_point,color, thickness)
cv2.imshow(window_name, image)

######################DRAW ARROW BETWEEN INTERFACE AND DUT#################################
start_point = (int(start_x_if - 15),int(start_y_if + 25))
# End coordinate
end_point = (int(start_x_dut - 15),int(start_y_dut - 5))
color = (0, 0, 0)
thickness = 3
# Using cv2.arrowedLine() method
image = cv2.arrowedLine(image, start_point, end_point,color, thickness)
cv2.imshow(window_name, image)
#tb_comps()
#read_file_draw()
######################DRAW ARROW BETWEEN DRIVER1 AND VIF#################################
start_point = (506,482)
# End coordinate
end_point = (506,547)
color = (0, 0, 0)
thickness = 3
```



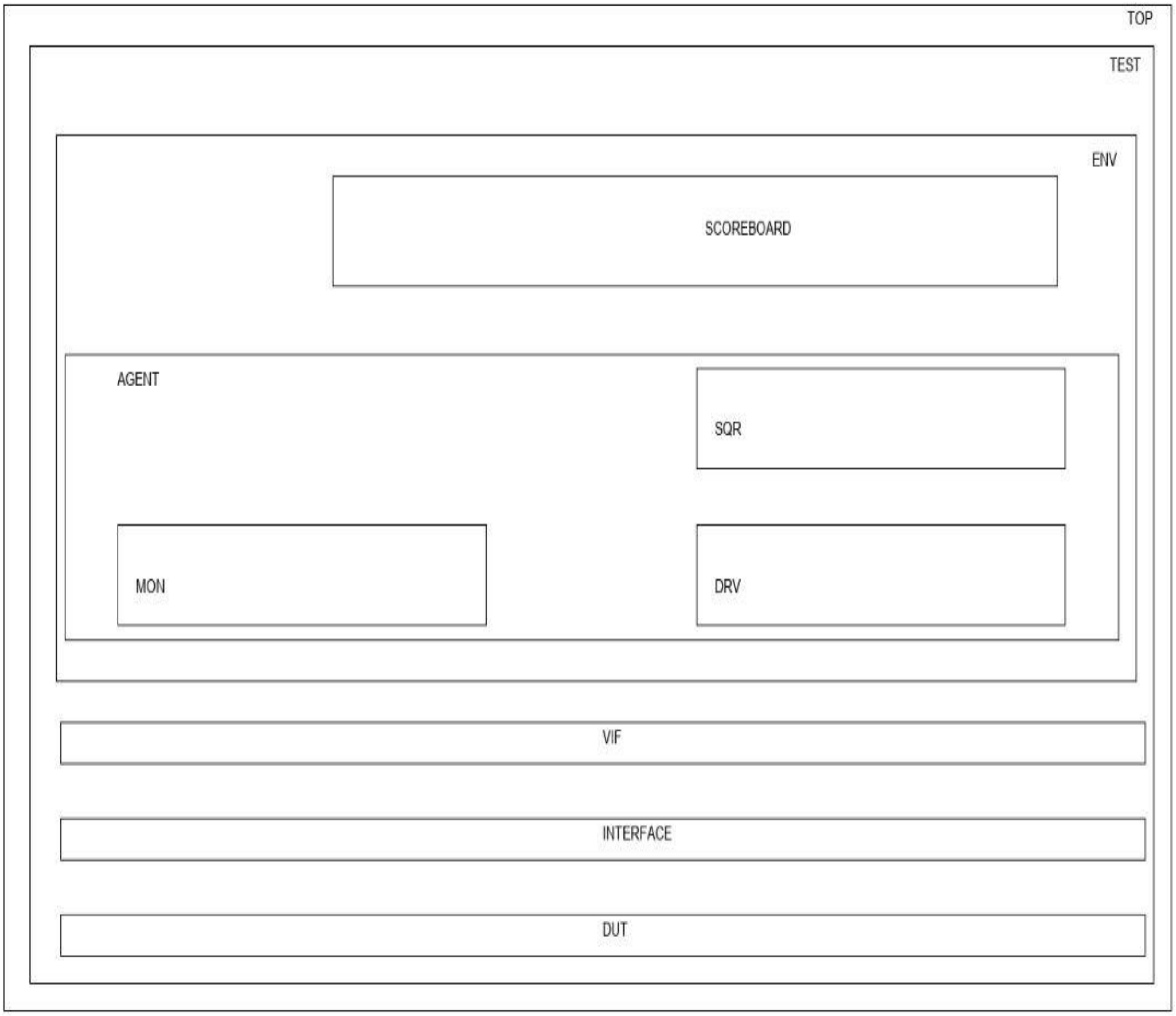
```
# Using cv2.arrowedLine() method
image = cv2.arrowedLine(image, start_point, end_point,color, thickness)
#tb_comps()
#read_file_draw()
cv2.imshow(window_name, image)
######################DRAW ARROW BETWEEN DRIVER2 AND VIF#################################
start_point = (836,482)
# End coordinate
end_point = (836,547)
color = (0, 0, 0)
thickness = 3
# Using cv2.arrowedLine() method
image = cv2.arrowedLine(image, start_point, end_point,color, thickness)
#tb_comps()
#read_file_draw()
cv2.imshow(window_name, image)
######################DRAW ARROW BETWEEN DRIVER2 AND VIF#################################
start_point = (236,482)
# End coordinate
end_point = (236,547)
color = (0, 0, 0)
thickness = 3
# Using cv2.arrowedLine() method
image = cv2.arrowedLine(image, start_point, end_point,color, thickness)
#tb_comps()
#read_file_draw()
cv2.imshow(window_name, image)
cv2.imwrite("E:\Python_task\\tb_arch.jpg",image)
doc.add_picture('E:\Python_task\\tb_arch.jpg')
doc.save('E:\Python_task\\pattern_printing_ex.docx')
~
```

Results of Create uvm_tb_name_image_file_generator:



```
enter path of TB directory: E:\uvm_tb_arch_doc_py-main\dummy_tb
keyword  :  name
uvm_test  :  mem_model_test
uvm_env  :  mem_model_env
uvm_scoreboard  :  mem_scoreboard
uvm_agent  :  mem_agent
uvm_driver  :  mem_driver
uvm_monitor  :  mem_monitor
uvm_sequencer  :  mem_sequencer
uvm_sequence_item  :  mem_seq_item
interface  :  instance,
2
number of agents : 2
enter a font_value:14
value of font_sixe 14
Width & HEIGHT 1366 768
Dimensions are %0d %0d %0d %0d (1356, 25) (25, 758)
Dimensions are %0d %0d %0d %0d (1336, 55) (55, 738)
Dimensions are %0d %0d %0d %0d (1316, 120) (85, 518)
1316
Navn_ag_cnt
1
VALUE OF X0 IS 0
VALUE OF X1 IS 95
VALUE OF X0 IS IF AGENT==1 1296.0
VALUE OF X0 LATER IS 1296.0
```

Results of Create uvm_tb_name_image_file_generator:


TOP
TEST
ENV
SCOREBOARD
AGENT
SQR
MON
DRV
VIF
INTERFACE
DUT