1.Snake game

import pygame

import time

snake game

pygame.init()

white = (255, 255, 255)

black = (0, 0, 0)

red = (255, 0, 0)

dis\_width = 800

dis\_height = 600

dis = pygame.display.set\_mode((dis\_width, dis\_width))

pygame.display.set\_caption('Snake Game by Subin MD')

game\_over = False

x1 = dis\_width / 2

y1 = dis\_height / 2

snake\_block = 10

x1\_change = 0

y1\_change = 0

clock = pygame.time.Clock()

snake\_speed = 30

font\_style = pygame.font.SysFont(None, 50)

def message(msg, color):

mesg = font\_style.render(msg, True, color)

dis.blit(mesg, [dis\_width / 2, dis\_height / 2])

while not game\_over:

for event in pygame.event.get():

if event.type == pygame.QUIT:

game\_over = True

if event.type == pygame.KEYDOWN:

if event.key == pygame.K\_LEFT:

x1\_change = -snake\_block

y1\_change = 0

elif event.key == pygame.K\_RIGHT:

x1\_change = snake\_block

y1\_change = 0

elif event.key == pygame.K\_UP:

y1\_change = -snake\_block

x1\_change = 0

elif event.key == pygame.K\_DOWN:

y1\_change = snake\_block

x1\_change = 0

if x1 >= dis\_width or x1 < 0 or y1 >= dis\_height or y1 < 0:

game\_over = True

x1 += x1\_change

y1 += y1\_change

dis.fill(white)

pygame.draw.rect(dis, black, [x1, y1, snake\_block, snake\_block])

pygame.display.update()

clock.tick(snake\_speed)

message("You lost", red)

pygame.display.update()

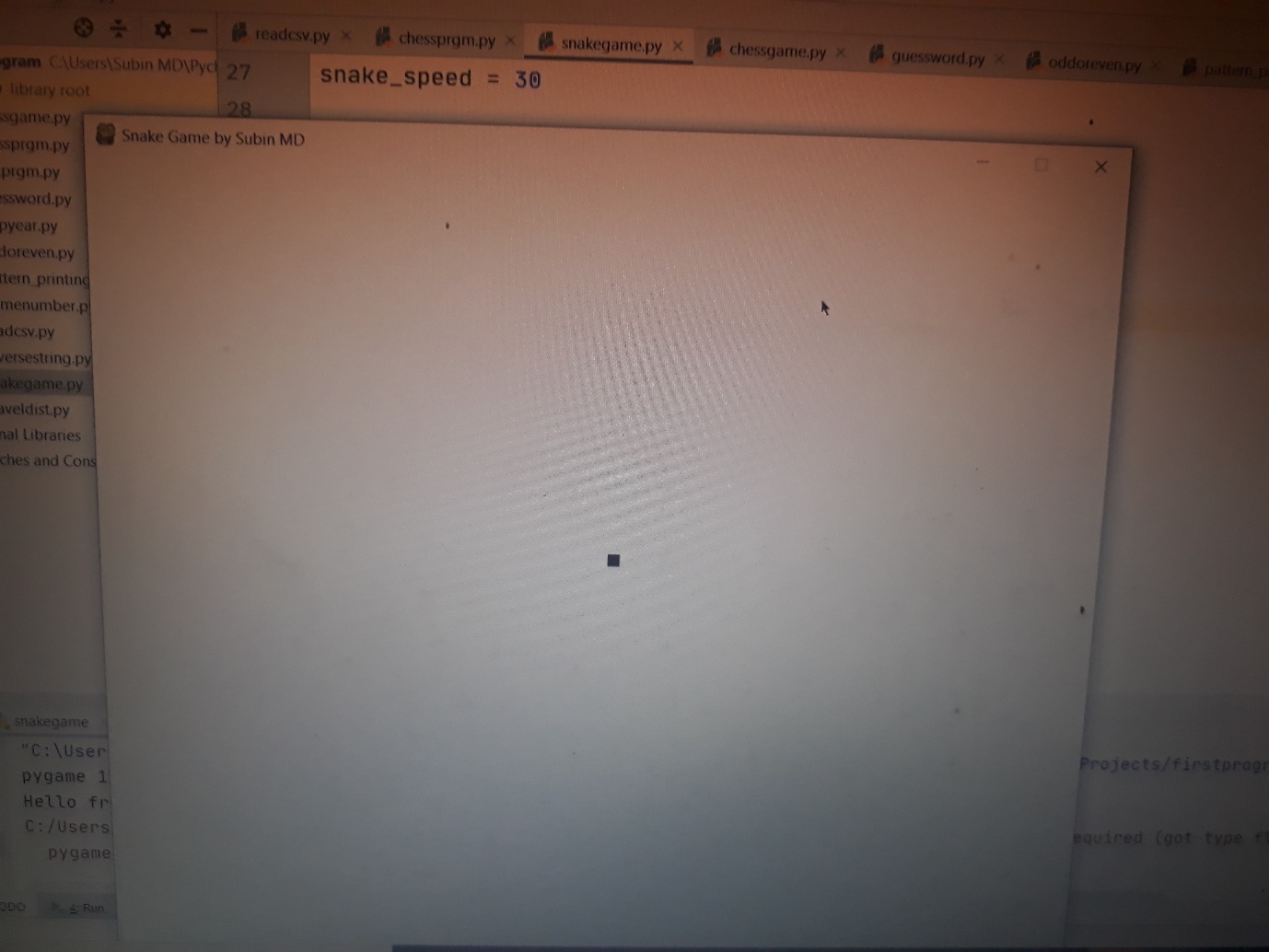
time.sleep(2)

pygame.quit()

quit()

output

starts from down arrow



2.

Alarm clock

**import** time  
  
  
  
Set\_Alarm = input(**"Set the alarm time(Hr):"**)  
print(**"You will have to wait until time reaches..!!!"**)  
  
  
  
Actual\_Time = time.strftime(**"%I:%M:%S"**)  
  
  
**while** (Actual\_Time != Set\_Alarm):  
 print  
 **"The time is "** + Actual\_Time  
 Actual\_Time = time.strftime(**"%I:%M:%S"**)  
 time.sleep(1)

output

Set the alarm time(Hr):2

You will have to wait until time reaches..!!!

3.

Web development(responsive page)

* pip install django
* pip install djangorestframework

from django.shortcuts import render

from django.http import Http404

from rest\_framework.views import APIView

from rest\_framework.decorators import api\_view

from rest\_framework.response import Response

from rest\_framework import status

from django.http import JsonResponse

from django.core import serializers

from django.conf import settings

import json

# Create your views here.

@api\_view(["POST"])

def IdealWeight(heightdata):

try:

height=json.loads(heightdata.body)

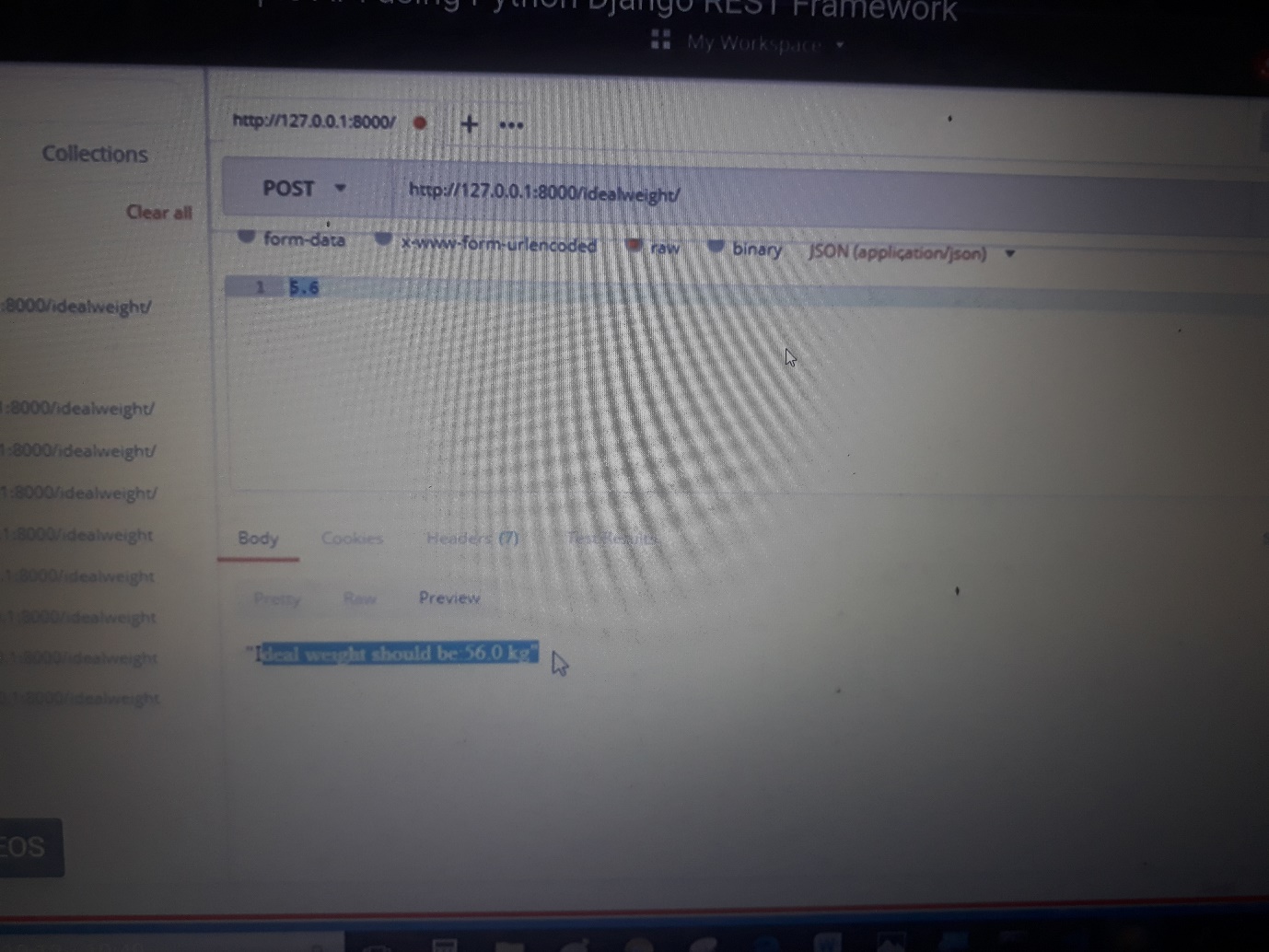
weight=str(height\*10)

return JsonResponse("Ideal weight should be:"+weight+" kg",safe=False)

except ValueError as e:

return Response(e.args[0],status.HTTP\_400\_BAD\_REQUEST)

output



4.

Sudoku game

**class** SudokuBoard(object):  
 *"""  
 Sudoku Board representation  
 """* **def** \_\_init\_\_(self, board\_file):  
 self.board = self.\_\_create\_board(board\_file)  
  
 **def** \_\_create\_board(self, board\_file):  
 board = []  
 **for** line **in** board\_file:  
 line = line.strip()  
 **if** len(line) != 9:  
 **raise** SudokuError(  
 **"Each line in the sudoku puzzle must be 9 chars long."** )  
 board.append([])  
  
 **for** c **in** line:  
 **if not** c.isdigit():  
 **raise** SudokuError(  
 **"Valid characters for a sudoku puzzle must be in 0-9"** )  
 board[-1].append(int(c))  
  
 **if** len(board) != 9:  
 **raise** SudokuError(**"Each sudoku puzzle must be 9 lines long"**)  
 **return** board  
  
  
**class** SudokuGame(object):  
 *"""  
 A Sudoku game, in charge of storing the state of the board and checking  
 whether the puzzle is completed.  
 """* **def** \_\_init\_\_(self, board\_file):  
 self.board\_file = board\_file  
 self.start\_puzzle = SudokuBoard(board\_file).board  
  
 **def** start(self):  
 self.game\_over = **False** self.puzzle = []  
 **for** i **in** xrange(9):  
 self.puzzle.append([])  
 **for** j **in** xrange(9):  
 self.puzzle[i].append(self.start\_puzzle[i][j])  
  
 **def** check\_win(self):  
 **for** row **in** xrange(9):  
 **if not** self.\_\_check\_row(row):  
 **return False  
 for** column **in** xrange(9):  
 **if not** self.\_\_check\_column(column):  
 **return False  
 for** row **in** xrange(3):  
 **for** column **in** xrange(3):  
 **if not** self.\_\_check\_square(row, column):  
 **return False** self.game\_over = **True  
 return True  
  
 def** \_\_check\_block(self, block):  
 **return** set(block) == set(range(1, 10))  
  
 **def** \_\_check\_row(self, row):  
 **return** self.\_\_check\_block(self.puzzle[row])  
  
 **def** \_\_check\_column(self, column):  
 **return** self.\_\_check\_block(  
 [self.puzzle[row][column] **for** row **in** xrange(9)]  
 )  
  
 **def** \_\_check\_square(self, row, column):  
 **return** self.\_\_check\_block(  
 [  
 self.puzzle[r][c]  
 **for** r **in** xrange(row \* 3, (row + 1) \* 3)  
 **for** c **in** xrange(column \* 3, (column + 1) \* 3)  
 ]  
 )

4.

Tic tac toe

**import** numpy **as** np  
**import** random  
**from** time **import** sleep  
  
**def** create\_board():  
 **return** (np.array([[0, 0, 0],  
 [0, 0, 0],  
 [0, 0, 0]]))  
  
  
**def** possibilities(board):  
 l = []  
  
 **for** i **in** range(len(board)):  
 **for** j **in** range(len(board)):  
  
 **if** board[i][j] == 0:  
 l.append((i, j))  
 **return** (l)  
  
  
  
**def** random\_place(board, player):  
 selection = possibilities(board)  
 current\_loc = random.choice(selection)  
 board[current\_loc] = player  
 **return** (board)  
  
  
  
**def** row\_win(board, player):  
 **for** x **in** range(len(board)):  
 win = **True  
  
 for** y **in** range(len(board)):  
 **if** board[x, y] != player:  
 win = **False  
 continue  
  
 if** win == **True**:  
 **return** (win)  
 **return** (win)  
  
  
**def** col\_win(board, player):  
 **for** x **in** range(len(board)):  
 win = **True  
  
 for** y **in** range(len(board)):  
 **if** board[y][x] != player:  
 win = **False  
 continue  
  
 if** win == **True**:  
 **return** (win)  
 **return** (win)  
  
  
**def** diag\_win(board, player):  
 win = **True** y = 0  
 **for** x **in** range(len(board)):  
 **if** board[x, x] != player:  
 win = **False** win = **True  
 if** win:  
 **for** x **in** range(len(board)):  
 y = len(board) - 1 - x  
 **if** board[x, y] != player:  
 win = **False  
 return** win  
  
  
**def** evaluate(board):  
 winner = 0  
  
 **for** player **in** [1, 2]:  
 **if** (row\_win(board, player) **or** col\_win(board, player) **or** diag\_win(board, player)):  
 winner = player  
  
 **if** np.all(board != 0) **and** winner == 0:  
 winner = -1  
 **return** winner  
  
  
**def** play\_game():  
 board, winner, counter = create\_board(), 0, 1  
 print(board)  
 sleep(2)  
  
 **while** winner == 0:  
 **for** player **in** [1, 2]:  
 board = random\_place(board, player)  
 print(**"Board after "** + str(counter) + **" move"**)  
 print(board)  
 sleep(2)  
 counter += 1  
 winner = evaluate(board)  
 **if** winner != 0:  
 **break  
 return** (winner)  
  
  
  
print(**"Winner is: "** + str(play\_game()))

output

[[0 0 0]

[0 0 0]

[0 0 0]]

Board after 1 move

[[0 1 0]

[0 0 0]

[0 0 0]]

Board after 2 move

[[0 1 0]

[0 2 0]

[0 0 0]]

Board after 3 move

[[0 1 0]

[1 2 0]

[0 0 0]]

Board after 4 move

[[0 1 0]

[1 2 2]

[0 0 0]]

Board after 5 move

[[0 1 0]

[1 2 2]

[0 0 1]]

Board after 6 move

[[0 1 0]

[1 2 2]

[2 0 1]]

Board after 7 move

[[0 1 1]

[1 2 2]

[2 0 1]]

Board after 8 move

[[2 1 1]

[1 2 2]

[2 0 1]]

Board after 9 move

[[2 1 1]

[1 2 2]

[2 1 1]]

Winner is: -1