# -------------------------------------------------------- 1 ----------------------------------------------------------

import time

import itertools

class TrafficLight:

\_\_color = [["red", [7, 31]], ["yellow", [2, 33]], ["green", [7, 32]], ["yellow", [2, 33]]]

def running(self):

for light in itertools.cycle(self.\_\_color):

print(f"\r\033[{light[1][1]}m\033[1m{light[0]}\033[0m", end="")

time.sleep(light[1][0])

traffic\_light\_1 = TrafficLight()

traffic\_light\_1.running()

# ------------------------------------------- вариант решения ---------------------------------------------------------

from time import sleep

class TrafficLight:

\_\_color = "Черный"

def running(self):

while True:

print("Trafficlight is red now")

sleep(7)

print("Trafficlight is yellow now")

sleep(2)

print("Trafficlight is green now")

sleep(7)

print("Trafficlight is yellow now")

sleep(2)

trafficlight = TrafficLight()

trafficlight.running()

# ------------------------------------------- вариант решения ---------------------------------------------------------

import time

import itertools

class TrafficLight:

\_\_color = [["red", [7, 31]], ["yellow", [2, 33]], ["green", [7, 32]], ["yellow", [2, 33]]]

def \_\_init\_\_(self, light\_list):

self.light\_list = light\_list

def running(self):

if len([i for i in self.light\_list if i in ["red", "yellow", "green"]]) >= 3:

for light in itertools.cycle(self.\_\_color):

print(f"\r\033[{light[1][1]}m\033[1m{light[0]}\033[0m", end="")

time.sleep(light[1][0])

else:

print("Your color list is incorrect.")

trafficlight\_1 = TrafficLight(["lilac", "green", "lime", "white", "black", "yellow"])

trafficlight\_1.running()

# ------------------------------------------- вариант решения ---------------------------------------------------------

from time import sleep

from itertools import cycle

class TrafficLight:

# \_\_color = [[' RED ', 'YELLOW', 'GREEN'], [7, 2, 4],

\_\_color = [" ", [7, 2, 4], ["\033[41m\033[1m", "\033[43m\033[1m", "\033[42m\033[1m"]]

def running(self):

col\_lst = ["", ""]

for n in cycle((0, 1, 2)):

col\_lst[1] = self.\_\_color[2][n]

print(f"\r({col\_lst[int(n == 0)]}{self.\_\_color[0]}\033[0m)"

f"({col\_lst[int(n == 1)]}{self.\_\_color[0]}\033[0m)"

f"({col\_lst[int(n == 2)]}{self.\_\_color[0]}\033[0m)", end='')

sleep(self.\_\_color[1][n])

tr\_light = TrafficLight()

tr\_light.running()

# ------------------------------------------- вариант решения ---------------------------------------------------------

from time import sleep

import colorama

import sys

colorama.init()

class TrafficLight:

# Запуск в терминале

def running(self):

while True:

print(f'\r\033[31m{chr(11035)} ', end='')

sleep(4)

print(f'\r\033[30m{chr(11035)} ')

print(f'\r\033[33m{chr(11035)} ', end='')

sleep(2)

print(f'\r\033[30m{chr(11035)} ')

print(f'\r\033[32m{chr(11035)} ', end='')

sleep(4)

print(f'\r\033[30m{chr(11035)} ', end='')

sys.stdout.write('\r\x1b[2A')

trafficLight = TrafficLight()

trafficLight.running()

# ------------------------------------------- вариант решения ---------------------------------------------------------

import tkinter as tk

from PIL import ImageTk, Image

from itertools import cycle

class TrafficLight:

def \_\_init\_\_(self, working\_algorithm):

self.sec\_count = 0

self.working\_algorithm = working\_algorithm

self.img\_dict = {'red': Image.open('pic/red.jpg').resize((250, 350)),

'yellow': Image.open('pic/yellow.jpg').resize((250, 350)),

'green': Image.open('pic/green.jpg').resize((250, 350)),

'red+yellow': Image.open('pic/red\_yellow.jpg').resize((250, 350)),

'off': Image.open('pic/off.jpg').resize((250, 350))}

self.iterator = cycle(self.working\_algorithm)

self.cur\_state = next(self.iterator)

self.root = tk.Tk()

self.root.attributes("-topmost", True)

self.root.geometry('300x400')

image = ImageTk.PhotoImage(self.img\_dict[self.cur\_state[0]])

self.label = tk.Label(image=image)

self.label.pack(expand=1, fill=tk.Y, side=tk.TOP)

self.running()

self.root.mainloop()

def running(self):

self.sec\_count += 1

if self.sec\_count == self.cur\_state[1]:

self.cur\_state = next(self.iterator)

self.sec\_count = 0

cur\_img = ImageTk.PhotoImage(self.img\_dict[self.cur\_state[0]])

self.label.configure(image=cur\_img)

self.label.photo\_ref = cur\_img

self.label.pack()

self.root.after(100, self.running)

mode = input("Enter traffic-light mode (0 - simple, 1-advanced):")

if mode == '0':

app = TrafficLight([('red', 70), ('yellow', 20), ('green', 50), ('yellow', 20)])

elif mode == '1':

app = TrafficLight([('red', 70),

('red+yellow', 20),

('green', 50),

('off', 5),

('green', 5),

('off', 5),

('green', 5),

('off', 5),

('green', 5),

('yellow', 20)])

else:

print('Wrong choice!')

# -------------------------------------------------------- 2 ----------------------------------------------------------

class Road:

def \_\_init\_\_(self, length, width):

self.\_length = length

self.\_width = width

def get\_full\_profit(self, weight=25, thickness=5):

return f"{self.\_length} м \* {self.\_width} м \* {weight} кг \* {thickness} см =" \

f" {(self.\_length \* self.\_width \* weight \* thickness) / 1000} т"

road\_1 = Road(5000, 20)

print(road\_1.get\_full\_profit())

# ------------------------------------------- вариант решения ---------------------------------------------------------

class Road:

def \_\_init\_\_(self, le, wi):

self.\_length = le

self.\_width = wi

def calc(self, weight=25, thickness=5):

print(f"Масса асфальта - {self.\_length \* self.\_width \* weight \* thickness / 1000} тонн")

def main():

while True:

try:

road\_1 = Road(int(input("Enter width of road in m: ")), int(input("Enter lenght of road in m: ")))

road\_1.calc()

break

except ValueError:

print("Only integer!")

# -------------------------------------------------------- 3 ----------------------------------------------------------

class Worker:

def \_\_init\_\_(self, name, surname, position, wage, bonus):

self.name = name

self.surname = surname

self.position = position

self.\_income = {"profit": wage, "bonus": bonus}

class Position(Worker):

def get\_full\_name(self):

return f"{self.name} {self.surname}"

def get\_full\_profit(self):

return f"{sum(self.\_income.values())}"

meneger = Position("Dorian", "Grey", "СEO", 500000, 125000)

print(meneger.get\_full\_name())

print(meneger.position)

print(meneger.get\_full\_profit())

# -------------------------------------------------------- 4 ----------------------------------------------------------

from random import choice

class Car:

""" Main car """

direction = ["🡡 north", "northeast 🡥", "east 🡢", "southeast 🡦",

"south 🡣", "🡧 southwest", "🡠 west", "🡤 northwest"]

def \_\_init\_\_(self, n, c, s, p=False):

self.name = n

self.color = c

self.speed = s

self.is\_police = p

print(f'New car: {n} has a color: {c}.\nIs the car a police car? {p}')

def go(self):

print(f'{self.name}: Car went.')

def stop(self):

print(f'{self.name}: Сar stopped!')

def turn(self):

print(f'{self.name}: Car turned {choice(self.direction)}.')

def show\_speed(self):

return f'{self.name}: Car speed: {self.speed}.'

class TownCar(Car):

""" City car """

def show\_speed(self):

return f'{self.name}: Car speed: {self.speed}. Speeding!' if self.speed > 60 else super().show\_speed()

class WorkCar(Car):

""" Cargo truck """

def show\_speed(self):

return f'{self.name}: Car speed: {self.speed}. Speeding!' if self.speed > 40 else super().show\_speed()

class SportCar(Car):

""" Sports Car """

class PoliceCar(Car):

""" Patrol car """

def \_\_init\_\_(self, n, c, s):

super().\_\_init\_\_(n, c, s, p=True)

police\_car = PoliceCar('"Полицайка"', 'белый', 80)

work\_car = WorkCar('"Грузовичок"', 'хаки', 40)

sport\_car = SportCar('"Спортивка"', 'красный', 120)

town\_car = TownCar('"Малютка"', 'жёлтый', 65)

list\_of\_cars = [police\_car, work\_car, sport\_car, town\_car]

for i in list\_of\_cars:

i.go()

print(i.show\_speed())

i.turn()

i.stop()

print()

# -------------------------------------------------------- 5 ----------------------------------------------------------

class Stationery:

def \_\_init\_\_(self, title="Something that can draw"):

self.title = title

def draw(self):

print(f"Just start drawing! {self.title}))")

class Pen(Stationery):

def draw(self):

print(f"Start drawing with {self.title} pen!")

class Pencil(Stationery):

def draw(self):

print(f"Start drawing with {self.title} pencil!")

class Marker(Stationery):

def draw(self):

print(f"Start drawing with {self.title} marker!")

stat = Stationery()

pen = Pen("Parker")

pencil = Pencil("Faber-Castell")

marker = Marker("COPIC")

office\_supplies = [stat, pen, pencil, marker]

for i in office\_supplies:

i.draw()