1. import re

class Smoothie:

ingredients\_price = {

'Strawberries':1.50,

'Banana':0.50,

'Mango':2.50,

'Blueberries':1.00,

'Raspberries':1.00,

'Apple':1.75,

'Pineapple':3.50

}

def \_\_init\_\_(self,ingredients):

self.ingredients = ingredients

self.cost = 0

def get\_cost(self):

for ele in self.ingredients:

if ele in Smoothie.ingredients\_price:

self.cost += round(Smoothie.ingredients\_price.get(ele,0),2)

return '$'+str(self.cost)

def get\_price(self):

self.price = round((self.cost\*1.5)+(self.cost),2)

return '$'+str(self.price)

def get\_name(self):

self.name = re.sub('berries','berry',' '.join(sorted(self.ingredients)))

self.name = self.name+' Smoothie' if len(self.ingredients) == 1 else self.name+' Fusion'

return self.name

s1 = Smoothie(["Banana"])

print(f's1.ingredients ➞ {s1.ingredients}')

print(f's1.get\_cost() ➞ "{s1.get\_cost()}"')

print(f's1.get\_price() ➞ "{s1.get\_price()}"')

print(f's1.get\_name() ➞ "{s1.get\_name()}"')

s2 = Smoothie(["Raspberries", "Strawberries", "Blueberries"])

print(f's2.ingredients ➞ {s2.ingredients}')

print(f's2.get\_cost() ➞ "{s2.get\_cost()}"')

print(f's2.get\_price() ➞ "{s2.get\_price()}"')

print(f's2.get\_name() ➞ "{s2.get\_name()}"')

Output:

s1.ingredients ➞ ['Banana']

s1.get\_cost() ➞ "$0.5"

s1.get\_price() ➞ "$1.25"

s1.get\_name() ➞ "Banana Smoothie"

s2.ingredients ➞ ['Raspberries', 'Strawberries', 'Blueberries']

s2.get\_cost() ➞ "$3.5"

s2.get\_price() ➞ "$8.75"

s2.get\_name() ➞ "Blueberry Raspberry Strawberry Fusion"

1. class Testpaper:

def \_\_init\_\_(self,subject,markscheme,pass\_mark):

self.subject = subject

self.markscheme = markscheme

self.pass\_mark = pass\_mark

class Student:

def \_\_init\_\_(self):

self.tests\_taken = "No tests taken"

def take\_test(self,paper\_name,student\_response):

correct\_responses = 0

for ele in range(len(paper\_name.markscheme)):

if paper\_name.markscheme[ele] == student\_response[ele]:

correct\_responses +=1

ach\_pass\_mark = int(correct\_responses/len(paper\_name.markscheme)\*100)

needed\_pass\_mark = int(paper\_name.pass\_mark.split("%")[0])

if self.tests\_taken == 'No tests taken':

self.tests\_taken = {}

if ach\_pass\_mark >= needed\_pass\_mark:

self.tests\_taken[paper\_name.subject] = f"Passed! ({str(ach\_pass\_mark)}%)"

else:

self.tests\_taken[paper\_name.subject] = f'Failed! ({str(ach\_pass\_mark)}%)'

paper1 = Testpaper("Maths", ["1A", "2C", "3D", "4A", "5A"], "60%")

paper2 = Testpaper("Chemistry", ["1C", "2C", "3D", "4A"], "75%")

paper3 = Testpaper("Computing", ["1D", "2C", "3C", "4B", "5D", "6C", "7A"], "75%")

student1 = Student()

student2 = Student()

print(f'student1.tests\_taken ➞ "{student1.tests\_taken}"')

student1.take\_test(paper1, ["1A", "2D", "3D", "4A", "5A"])

print(f'student1.tests\_taken ➞ {student1.tests\_taken}')

student2.take\_test(paper2, ["1C", "2D", "3A", "4C"])

student2.take\_test(paper3, ["1A", "2C", "3A", "4C", "5D", "6C", "7B"])

print(f'student2.tests\_taken ➞ {student2.tests\_taken}')

Output:

student1.tests\_taken ➞ "No tests taken"

student1.tests\_taken ➞ {'Maths': 'Passed! (80%)'}

student2.tests\_taken ➞ {'Chemistry': 'Failed! (25%)', 'Computing': 'Failed! (42%)'}

1. class Train:

def \_\_init\_\_(self,destination\_list,expected\_time):

self.destination\_list = destination\_list

self.expected\_time = expected\_time

def manage\_delays(train\_obj,destination\_point,delay):

if destination\_point in train\_obj.destination\_list:

ex\_time = train\_obj.expected\_time.split(":")

new\_ex\_time = str((int(ex\_time[0])\*60+int(ex\_time[1])+int(delay))//60)+':'+str((int(ex\_time[0])\*60+int(ex\_time[1])+int(delay))%60)

train\_obj.expected\_time = new\_ex\_time

trains = [

Train(["Townsville", "Suburbia", "Urbantska"], "13:04"),

Train(["Farmsdale", "Suburbia", "Lakeside Valley"], "13:20"),

Train(["Suburbia", "Townsville", "Lakeside Valley"], "13:22")

]

for t in trains:

manage\_delays(t, "Lakeside Valley", 60)

print(f'trains[0].expected\_time ➞ "{trains[0].expected\_time}"')

print(f'trains[1].expected\_time ➞ "{trains[1].expected\_time}"')

print(f'trains[2].expected\_time ➞ "{trains[2].expected\_time}"')

Output:

trains[0].expected\_time ➞ "13:04"

trains[1].expected\_time ➞ "14:20"

trains[2].expected\_time ➞ "14:22"

1. class minecraft:

def \_\_init\_\_(self,tracks):

self.tracks = tracks

def mine\_run(tracks):

mine\_vel = 0

output = 0

for ele in range(len(tracks)):

if tracks[ele] == '-->':

if mine\_vel+2.67 >= 8: mine\_vel = 8 # resetted to max mine\_val

else: mine\_vel +=2.67

elif tracks[ele] == '<-->': continue

elif tracks[ele] == '<--':

if mine\_vel-2.67 <= 0: output = True if ele == len(tracks)-1 else ele; break

else: mine\_vel -=2.67

else:

if mine\_vel-1 <= 0: output = True if ele == len(tracks)-1 else ele; break

else: mine\_vel -=1

return output

mine\_track\_1 = minecraft(["-->", "-->", "-->", "<--", "<--", "<--"])

print(f'mine\_run({mine\_track\_1.tracks}) ➞ {mine\_run(mine\_track\_1.tracks)}')

mine\_track\_2 = minecraft(["-->", "<--", "-->", "-->", "<-->", "---"])

print(f'mine\_run({mine\_track\_2.tracks}) ➞ {mine\_run(mine\_track\_2.tracks)}')

Output:

mine\_run(['-->', '-->', '-->', '<--', '<--', '<--']) ➞ True

mine\_run(['-->', '<--', '-->', '-->', '<-->', '---']) ➞ 1

1. def intersecting(rectangle1, rectangle2):

output = False

if (rectangle2.y-rectangle1.y == rectangle1.height) or (rectangle2.y-rectangle1.y + rectangle1.height == rectangle2.height):

output = True

print(f'intersecting{rectangle1.\_\_dict\_\_.values(),rectangle2.\_\_dict\_\_.values()} ➞ {output}')

class Rectangle:

def \_\_init\_\_(self, x, y, width, height):

self.x = x

self.y = y

self.width = width

self.height = height

a = Rectangle(10, 20, 100, 20)

b = Rectangle(10, 40, 15, 20)

c = Rectangle(50, 50, 20, 30)

intersecting(a, b)

intersecting(a, c)

intersecting(b, c)

Output:

intersecting(dict\_values([10, 20, 100, 20]), dict\_values([10, 40, 15, 20])) ➞ True

intersecting(dict\_values([10, 20, 100, 20]), dict\_values([50, 50, 20, 30])) ➞ False

intersecting(dict\_values([10, 40, 15, 20]), dict\_values([50, 50, 20, 30])) ➞ True