THE CRC ROBOTICS
SENIOR PROGRAMMING PROBLEMS

PRELIMINARY 3

MODUEL
2026

presented by
FTAI AVIATION

A program of
AEST EAST

Version 1.0

A FEW NOTES

- The complete rules are in section 4 of the rulebook.
- You have until **Friday, January 23rd, 11:59 pm** to submit your code.
- Feel free to use the programming forum on the CRC discord to ask questions and discuss the problem with other teams. It is there for that purpose!
- **We are giving you quick and easy-to-use template files for your code and the tests. You are required to use them.**

USING THE TEMPLATE FILE

- The template tests call the function to test, take the output and allow you to quickly check if your code works as intended. **All your code, except additional functions you create, should be written in the function of the part of the problem you are solving.**
- Points given in the document are indications of how difficult the section is and how many points you will get if you complete it. This preliminary problem is going to be 2% of the main challenge towards the global score of the programming competition and for more points related information consult the rulebook.

STRUCTURE

Every problem contains a small introduction like this about the basics of the problem and what is required to solve it.

Input and output specification:

Contains the inputs and their format, and which outputs the code is required to produce and in what format they shall be.

Sample input and output:

Contains a sample input, sometimes containing sub examples in the sample input, and what your program should return as an output.

Explanation of the first output:

Explains briefly the logic that was used to reach the first output, given the first input.

Abeilles and bees!

In this third prelim problem we'll explore the world of abeilles and bees! But, differently than the problem for platypusses, we will not focus on learning some new characteristics about bees but rather explore all the puns that can be made using abeille and bee!

Part 1: Fizz-Buzz-Bees (10 points)

The game fizz-buzz is a simple classic game that can be played in a car ride or around a campfire. But today we bring a slight update making it fizz-buzz-bees.

In the game fizz-buzz, you count numbers, but if you hit a number divisible by 3 you have to say Fizz instead of the number. If you reach a number divisible by 5 you have to say Buzz instead of the number. We are adding a third rule, if you reach a number divisible by 7 you have to say Bees instead of the number.

When you hit a number that is divisible by 3 and 5, you have to say FizzBuzz. The same logic will apply if you have a number divisible by 3, 5 and 7, that would be replaced with FizzBuzzBees.

Here is an example of the sequence starting from 1 to 10:
[“1”, “2”, “Fizz”, “4”, “Buzz”, “Fizz”, “Bees”, “8”, “Fizz”, “Buzz”]

Input and output specification:

You will receive as input an *int* that is the first number in the sequence that needs to be included and you will receive a second *int* that will be the last number in the sequence that you will have to include.

You will return a *list* of *string* that is either the number as a *string* or the FizzBuzzBees

Sample input:

start = 9
end=15

start=100
end=110

start=20
end=42

Sample output

```
['Fizz', 'Buzz', '11', 'Fizz', '13', 'Bees', 'FizzBuzz']

['Buzz', '101', 'Fizz', '103', '104', 'FizzBuzzBees', '106', '107', 'Fizz', '109', 'Buzz']

['Buzz', 'FizzBees', '22', '23', 'Fizz', 'Buzz', '26', 'Fizz', 'Bees', '29', 'FizzBuzz', '31', '32',
'Fizz', '34', 'BuzzBees', 'Fizz', '37', '38', 'Fizz', 'Buzz', '41', 'FizzBees']
```

Explanation of the first output:

In the first example, we count from 9 included to 15 included. So we can start with 9 that is divisible by only 3 so the number needs to be replaced by “Fizz”. For the number 10, we see that it is only divisible by 5 so we replace it by “Buzz”. Then for 11, since it is not divisible by 3, 5 or 7 we output the number 11 in a string: “11”. Then for 12 it is divisible by 3 so we have “Fizz”, for 13 it is not divisible by 3, 5 or 7 so we output “13”. For 14 it is divisible by 7 so we replace for “Bees”. Finally, for 15, it is divisible by both 3 and 5 so we output “FizzBuzz”. This gives us the final answer:
['Fizz', 'Buzz', '11', 'Fizz', '13', 'Bees', 'FizzBuzz']

Part 2: Fris-bee (X points)

This morning as I was walking through the park I saw some bees playing frisbee. As any sane person would do seeing bees playing frisbee I was shocked to realise that I didn’t know…

What is the physics being a frisbee being thrown!!

So I decided to do some research and ended up finding this article from MIT (https://web.mit.edu/womens-ult/www/smite/frisbee_physics.pdf) that introduces all the core concepts from frisbee throw. We are going to base ourselves on that paper to make a program that will compute the path of the frisbee for the first 5 seconds of flight.

Here are the core concepts of the flight of a frisbee:

There are 2 axes along which we are going to track the flight. The frist one is the horizontal distance from the throwing point (x axis) and the second axis is the distance of the frisbee from the ground (y axis).

There are 3 major forces that act on the frisbee. The first is the gravity that pull down on the y axis. The second force is the lift generated that pushes the frisbee up on the y axis. Finally, the last major force acting is the drag that slows down the frisbee on the x axis.

In our simulation, we will track the change in speed and position along the x and y axis every 0.01 seconds. We will do the tracking 100 times, so for a full second of flight. You will have to compute the position on the x and y axis with a precision of 3 significant digits, but make sure to keep all the precision possible throughout the computation.

In all the exemples, you will receive the initial speed in x and y also with the angle of attack from the frisbee. All other variables are constants and are already given to you in test files.

Before we start the simulation a few variables are needed to be computed. Those variables are the lift coefficient and the drag coefficient. Here are the formulas:

$$CL = CL0 + CLA * angle * PI/180$$
$$CD = CD0 + CDA * ((angle - ALPHA0) * PI/180)^2$$

We now have all variables ready for the simulation! We can now compute our 100 data points for the x and y axis along the flight path (keeping 3 significant digits) for every interval of 0.01 second for up to a second of flight.

For all iterations of 0.01 seconds, we<ll need to find the change of speed (delta) for the x and y axis. Once the deltas are computed, we can find the new speed in and x and y. With the updated speed, we can update the position and finally record our position in the flight! Once it is done, we can repeat the process for each of our 100 iterations!

Here are the formulas to compute the speed difference for the x and y axis for 0.01s:

$$\Delta vy = (AIRDENSITY * vx^2 * AREA * CL * MASS/2 + G) * \Delta t$$
$$\Delta vx = - AIRDENSITY * vx^2 * AREA * CD * \Delta t$$

$$vx = vx + \Delta vx$$
$$vy = vy + \Delta vy$$
$$x = x + vx * \Delta t$$
$$y = y + vy * \Delta t$$

One final rule for our simulation is that we need to stop the simulation once the frisbee touches the ground (y <= 0). We then need to stop adding more data points since the flight is completed. That means that all of our y coordinates will be above 0 (or 0 in case of some rounding).

Input and output specification:

For input, you will receive

- vx0: a *float* of the initial speed in the x axis
- vy0: a*float* of the initial speed on the y axis
- angle: a*float* of the angle of the frisbee

For output, you need to output a *list* of *tuples* that are the position along the x and y axis with a precision of 3 digits.

Sample input:

```
vx0 = 16
vy0 = 0
angle = 0

vx0 = 15
vy0 = 2
angle = 5
```

Sample output:

```
[(0.16, 0.999), (0.319, 0.997), (0.479, 0.994), (0.638, 0.99), (0.798,
0.986), (0.957, 0.98), (1.115, 0.973), (1.274, 0.965), (1.433, 0.957),
(1.591, 0.947), (1.749, 0.936), (1.907, 0.925), (2.065, 0.912), (2.223,
0.899), (2.38, 0.884), (2.538, 0.869), (2.695, 0.852), (2.852, 0.835),
(3.009, 0.817), (3.165, 0.797), (3.322, 0.777), (3.478, 0.756), (3.635,
0.733), (3.791, 0.71), (3.947, 0.686), (4.102, 0.661), (4.258, 0.635),
(4.414, 0.608), (4.569, 0.58), (4.724, 0.551), (4.879, 0.521), (5.034,
0.49), (5.188, 0.458), (5.343, 0.425), (5.497, 0.392), (5.652, 0.357),
(5.806, 0.321), (5.959, 0.284), (6.113, 0.247), (6.267, 0.208), (6.42,
0.168), (6.574, 0.128), (6.727, 0.086), (6.88, 0.044), (7.032, 0.0)]

[(0.15, 1.019), (0.299, 1.037), (0.449, 1.054), (0.598, 1.07), (0.747,
1.086), (0.895, 1.1), (1.044, 1.113), (1.192, 1.126), (1.34, 1.137),
(1.487, 1.148), (1.635, 1.157), (1.782, 1.166), (1.929, 1.173), (2.076,
1.18), (2.223, 1.186), (2.369, 1.191), (2.515, 1.195), (2.661, 1.197),
(2.807, 1.199), (2.952, 1.2), (3.098, 1.2), (3.243, 1.199), (3.388,
1.197), (3.532, 1.195), (3.677, 1.191), (3.821, 1.186), (3.965, 1.18),
(4.109, 1.174), (4.252, 1.166), (4.396, 1.158), (4.539, 1.148), (4.682,
1.138), (4.824, 1.126), (4.967, 1.114), (5.109, 1.101), (5.251, 1.086),
(5.393, 1.071), (5.535, 1.055), (5.676, 1.038), (5.818, 1.02), (5.959,
1.001), (6.1, 0.981), (6.24, 0.96), (6.381, 0.938), (6.521, 0.915),
(6.661, 0.891), (6.801, 0.866), (6.941, 0.841), (7.08, 0.814), (7.219,
0.786), (7.358, 0.758), (7.497, 0.728), (7.636, 0.698), (7.774, 0.666),
(7.912, 0.634), (8.051, 0.601), (8.188, 0.566), (8.326, 0.531), (8.464,
0.495), (8.601, 0.458), (8.738, 0.419), (8.875, 0.38), (9.012, 0.34),
(9.148, 0.299), (9.284, 0.257), (9.42, 0.214), (9.556, 0.171), (9.692,
0.126), (9.828, 0.08), (9.963, 0.033)]
```

Explanation of the first output:

Use the explanation and refer to the MIT reference if needed.

Part 3: Bee-thoven (X points)

Have you heard of the legendary pianist Bee-thoven? Apparently, he's making a buzz in the hive with his honeyed serenades.

But lately, Bee-thoven has hit a dead end and can no longer compose symphonies as brilliant as before. Your mission: find a **sequence of 4 measures** that satisfies all of his requirements and helps him regain his inspiration.

For each measure, Bee-thoven provides you with a list of note groups to choose from. Each group has several properties:

- **id**: the group's identifier
- **pitch**: the pitch of the measure (a number from 1 to 12)
- **duration**: the duration of the measure
- **type**: the category of the measure (0, 1, or 2)
- **higher**: whether the pitch of this group must be higher than that of the previous measure (0 or 1)
- **forbidden**: whether this group cannot follow a group of the specified type (-1 or the type to avoid)

Additional rules to follow:

- You must choose **exactly one group per measure**.
- Two consecutive groups **cannot have the same type**.
- The **sum of the durations** of the 4 measures must be **exactly 10**.
- The **higher** and **forbidden** constraints must be respected.

Input and output specification:

As input, you will receive a ***list* of 4 measures**. Each measure is itself a ***list*** of the note groups that can be chosen for that measure. Each group is represented by a ***tuple*** in the form : **`(id, pitch, duration, type, higher, forbidden).`**

As output, you must return a ***list* of IDs** corresponding to the correct sequence of groups that satisfies Bee-thoven's requirements.

Sample input:

```
1. [[(0, 5, 2, 1, 0, -1), (1, 3, 3, 0, 0, -1)],
    [(10, 4, 3, 1, 1, -1), (11, 7, 1, 2, 0, 0)],
    [(20, 3, 3, 1, 0, -1), (21, 8, 4, 2, 0, 1), (22, 6, 2, 0, 1, -1)],
    [(30, 9, 2, 2, 0, -1), (31, 4, 5, 0, 0, -1)]]

2. [[(0, 4, 3, 0, 0, -1), (1, 6, 4, 1, 0, -1)],
    [(10, 7, 2, 1, 0, -1), (11, 5, 3, 2, 0, -1)],
    [(20, 8, 3, 2, 0, -1), (21, 3, 4, 0, 0, -1)],
    [(30, 9, 2, 0, 0, -1), (31, 2, 5, 1, 0, -1)]]

3. [[(0, 6, 3, 1, 0, -1), (1, 3, 2, 0, 0, -1)],
    [(10, 5, 2, 2, 1, -1), (11, 4, 4, 1, 0, 0)],
    [(20, 7, 5, 0, 1, -1), (21, 2, 3, 2, 0, 1), (22, 9, 3, 1, 0, -1)],
    [(30, 4, 7, 0, 0, -1), (31, 10, 3, 2, 0, -1)]]
```

Sample output:

```
1. [1, 10, 22, 30]
2. [0, 10, 20, 30]
3. [1, 10, 22, 31]
```

Explanation of the first output:

For the **first measure**, both groups have no constraints, so either one can be chosen.
Possibilities: [0, …, …, …] or [1, …, …, …]

For the **second measure**, group 10 must have a pitch higher than the previous measure, so it can only follow group 1 (4 > 3). Group 11 cannot follow a group of type 0, so it can only follow group 0.
Possibilities: [0, 11, …, …] or [1, 10, …, …]

For the **third measure**, only group 22 can be placed after group 10, because group 20 is the same type (type 1), group 21 is forbidden after type 1, and group 22 has a pitch higher than the previous measure (6 > 4). By the same logic, only group 20 can be placed after group 11, because group 21 is the same type (type 2) and group 22 has a pitch lower than the previous measure (6 < 7).
Possibilities: [0, 11, 20, …] or [1, 10, 22, …]

For the **fourth measure**, the sum of durations must equal 10, and there is only one combination of groups that satisfies all rules and the total duration: **[1, 10, 22, 30]**.

Part 4: bee-p bee-p! (X points)

A bee needs to go home, but there are a lot of other bees that also need to get home. Help the bee get home the fastest way possible.

Input and output specification:

As an input you'll get
r = an *int* of the amount of possible paths

v = a *list* of *int* of the speed limit in m/h for each path

t = a *list* of *int* of the speed of the traffic in m/h

traffic_distance = a *list* of *int* of the length in meter of the traffic

d = a *list* of *int* the total distance per path in meter

As an output, you'll have to give the *int* of the number of the shorts path depending on the time.

Sample input:

r = 2
v = (50, 40)
t = (15, 25)
trafic_distance = (15, 4)
d = (30, 25)

r = 3
v = (60, 30, 25)
t = (20, 25, 24)
trafic_distance = (30, 40, 40)
d = (45, 50, 47)

r = 3
v = (30, 30, 30)
t = (13, 15, 12)
trafic_distance = (15, 12, 13)
d = (30, 30, 30)

Sample output:

```
2

1

2
```

Explanation of the first output:

The first path has 15 meters of traffic at 15m/h and 15 meters at 50m/h. That makes 1h in traffic and 18 minutes outside of it. The time is calculated by dividing the length by the speed and multiple by 60 to have the time in minutes. The second path has 4 meters at 25m/h that takes 9 minutes 36s and 21 meters at 40m/h which takes 31 minutes 30s. So the second path is faster!

Part 5: You won’t bee-lieve how much this hurts! (X points)

OUCH! A bee has stung you and you want to know what kind of anti venom / pain medicine to take to help with the pain and swelling of the sting. You will have to use the level of pain felt when you got stung, the geographical area where you were when it happened as well as the taxonomical family of the bee and the type of nest it came from. You will then have to decide what type of remedy to use to quell the pain and help with the inflammation. Here is the list of different types of bees that could have gotten you with their stinger:

Type of bee	Pain level	Geographic area	taxonomical family	Type of nest
sweat bee	4	africa, asia, europe	halictidae	ground, tree
killer bee	8	north america, south america, africa	apidae	ground, hole
carpenter bee	5	oceania, europe	apidae	dead wood, bamboo
honey bee	4	north america, south america, asia, europe	apidae	hole, artificial nest
bumblebee	2	north america, south america, asia, europe	apidae	hole, tree, garden
dark bee	8	europe, north america	apidae	tree
buckfast bee	6	europe, africa	apidae	artificial nest
mining bee	6	north america, africa, europe, asia	andrenidae	ground
mason bee	4	asia, oceania	megachilidae	rock, tree, bamboo
resin bee	5	europe	megachilidae	artificial nest, tree

The possible remedies are in the following list:

- epinephrine (sweat bee, carpenter bee, bumblebee, mining bee)
- antihistamine (killer bee, resin bee, dark bee, buckfast bee)
- ice (mason bee, honey bee, bumblebee)
- antiseptic cream (killer bee, dark bee, resin bee)

***The list of remedies is semi-representative of real life. Do not use this programming problem as a guide to take care of a bee sting.

Spécification d'entrée et de sortie:

As input, you will be given the level of pain felt, the geographical location where the sting happened, the taxonomical family of the bee and the type of nest it came from.
pain: int
region: str
family: str
nest_type: str
As output, you will give a *list[str]* with 1 to x remedies to use to help with the sting

Exemple d'entrée:

```
pain=4 geography=europe family=halictidae nest=ground
pain=2 geography=asia family=apidae nest=hole
pain=8 geography=africa family=apidae nest=ground
```

Exemple de sortie:

```
[epinephrine]
[epinephrine, ice]
[antihistamine, antiseptic cream]
```

Explication de la première sortie:

The only type of bee that fits all of the criteria that were given as input is the sweat bee and the remedy is epinephrine.