THE HONG KONG UNIVERSITY OF SCIENCE AND TECHNOLOGY

COMP1021 Introduction to Computer Science

Final Examination

12:30pm – 2:35pm
(2 hours and 5 minutes duration)

Name:	
Student id:	
Your lecture: ***L1***=*Monday 2pm/Friday 9:30am* ***L2***=*Monday/Wednesday 12nn* ***L3***=*Monday/Wednesday 9:30am*	*Write L1 or L2 or L3:*
Your lab: ***LA1***=*Wednesday 5:30pm* ***LA2***=*Tuesday 2:30pm* ***LA3***=*Friday 10:30am* ***LA4***=*Thursday 10:30am*	*Write LA1 or LA2 or LA3 or LA4:*

Instructions

- This is an open book, open notes examination
- Simple calculators are permitted; mobile phones, tablets and computers are not
- The highest possible mark is 100
- Read each question carefully before answering
- Write your answers clearly in the space provided **in this exam script**
- You need to return this exam script, all pages, for marking
- Be careful to use capital/small letters at the appropriate places
- Assume that the questions use the version of Python used on the course, Python 3
- Assume that the questions use the same modules used on the course

Q1) ***4 marks***

Here is a Python program.

You haven't learned .ontimer() and .update() in this semester and won't be able to answer this question.

```
import turtle
import random

turtle.bgcolor("black")
turtle.hideturtle()
turtle.up()
turtle.tracer(False)

colours = ["red", "orange", "yellow", "lightgreen", \
           "green", "cyan", "magenta"]

def disco():
    global colours

    turtle.clear()

    turtle.color(colours[0])
    for _ in range(50):
        turtle.goto(random.randint(-400, 400), \
                    random.randint(-400, 400))
        turtle.dot(random.randint(10, 30))
    turtle.update() # Update the display

    colours = colours[1: ] + colours[ :1]

def letsdance():
    disco()
    turtle.ontimer(letsrock, 500)

def letsrock():
    disco()
    turtle.ontimer(letsdance, 800)

letsrock()

turtle.done()
```

If you run the above program, what will be the colour of the dots inside the turtle window **exactly 3 seconds after the program starts**? Please write the colour name in the answer below.

If you think nothing is shown in the turtle window 3 seconds after the program starts, write ‘Nothing’.

Answer: ____

Q2) ***4 marks***

Here is a Python program.

```
import turtle

turtle.colormode(255)
turtle.setworldcoordinates(0, 0, 255, 255)
turtle.hideturtle()

def dye(x, y):
    turtle.bgcolor(int(x), int(y), int(max(x - y, 0)))

turtle.onscreenclick(dye)
turtle.done()
```

If you run the program and click on the **top-right hand corner** inside the turtle window, what will the background colour of the window become?

If you cannot see the exact answer, choose the closest one.

A) White   B) Yellow   C) Blue   D) Grey

Answer (A/B/C/D): ___________

Q3) ***4 marks***

A Python function called `recursion` has been created as shown below. The function recursively performs some calculations and returns the result based on a given **positive integer n**.

```
def recursion(n):
    if n == 1:
        return 1
    else:
        return recursion(n - 1) * n
```

This question is suitable for this semester after you learn recursion

What will be the value returned by the function if you run `recursion(3)`?

A) 1
B) 2
C) 3
D) 6

Answer (A/B/C/D): ___________

Q4) ***4 marks***

Four Python programs are shown below.

Three of them produce the same result (i.e. they print the same output). One of them produces a different result compared to the other three. Which one of the following programs produces a different result compared to the other three programs?

A)
```
mylist = [1, 2, 4, 8, 12]
result = []
while len(mylist):
    result.append(mylist.pop())
mylist = result
print(mylist)
```

B)
```
mylist = [1, 2, 4, 8, 12]
for i in range(len(mylist)):
    item = mylist[i]
    mylist[i] = mylist[-i-1]
    mylist[-i-1] = item
print(mylist)
```

C)
```
mylist = [1, 2, 4, 8, 12]
mylist = mylist[ : :-1]
print(mylist)
```

D)
```
mylist = [1, 2, 4, 8, 12]
mylist.reverse()
print(mylist)
```

Answer (A/B/C/D): ___________

Q5) ***4 marks***

The following Python program adds up **all odd numbers** from 1 to 5 and then shows the result on the screen.

```
def sum_odd_numbers(end):
    for number in range(1, end + 1):
        if number % 2 == 1:
            result = result + number

result = 0
sum_odd_numbers(5)
print(result)
```

Is the above program working correctly?

Answer (YES/NO): ___________

Q6) ***4 marks***

A student has created a Python program to help calculate the total number of credits completed so far in a semester.

```
total = 0
credits = 0

print("Please enter the credits of each course one by one.")
print()

while credits != "done":
    print("Enter the number of credits of your course")
    credits = input("(or enter 'done' to finish the program): ")
    if credits != "done":
        if type(credits) != int:
            print("Please give me an integer!")
        else:
            total += credits

print("The number of credits you have taken so far is:", total)
```

If you run the program, type `3`, press *Enter*, type `4`, press *Enter* and then type `done` and press *Enter*, **what is the number of credits you have taken so far**, according to the output of the program?

Answer: ____

Q7) ***4 marks***

Here is a Python function with two input values.

```
def find(number_list, n):
    answer = []
    for number in number_list:
        if not number % n:
            answer.append(number)
    return len(answer)
```

What will be the value returned by the function if you run `find([1, 3, 7], 3)`?

A) 1
B) 2
C) 3
D) 7

Answer (A/B/C/D): ____________

Q8) *4 marks*

In this question, you need to use the `find` function of a string. The function returns the **first position** of a string within another string. If the function cannot find the string, it will return -1. Here is an example of using the function in the shell.

```
>>> print("the cat in the hat".find("at"))
5
```

Here is a Python program which uses the `find` function.

```
def do(me, you):
    result = 0
    while you in me:
        me = me[me.find(you) + len(you): ]
        result += 1
    return result

print(do("A cat wearing a hat is sitting on a mat!", "at"))
```

This question is suitable for this semester, although students taking this question in Spring 2017 had more experience with text handling

If you run the above program, what will be the output of the program?

Answer: ____

Q9) *4 marks*

Here is a Python program.

```
size = int(input("What is the size? "))
for i in range(size, 0, -1):
    print(" " * (i - 1) + "X" * ((size - i) * 2 + 1))
for i in range(2, size + 1):
    print(" " * (i - 1) + "X" * ((size - i) * 2 + 1))
```

If you run the above program, type `3` and then press Enter, what will be the output of the program?

You need to draw the output of the program in the space below by filling an **X** in all the appropriate boxes. Each box represents one character in the output. The output of the program starts from the box at the top-left hand corner. You may or may not not need to use up all the rows and columns provided.

Answer:

Q10) ***4 marks***

Here is a Python program.

```
black_rainstorm = True
amber_rainstorm = False
early = False
enough_marks = True
need_to_graduate = True

result = not black_rainstorm and not amber_rainstorm and not early or \
         enough_marks or need_to_graduate

print(result)
```

When the above program is executed, what **exactly** does the program print?

Your answer is one word. Write exactly what the program prints.

Answer: ____

Q11) ***4 marks***

Here is a Python program.

```
#       01234567890123456789012345678901234
text = "little cove is the best coffee shop"
#       54321098765432109876543210987654321

print( text[15:19] + text[27:30] + text[-4:-5:-1] )
```

What do you see when the above program is run?

If the output has a space then make sure you have a clear gap/space when you write your answer.

Answer: ____

Q12) ***8 marks*** Here is a Python program.

```
row=1
col=1
finished=False
locations = [ ["scary forest", "twisty path", "deep dark cave"],
    ["dirty kitchen", "dusty sitting room", "pretty garden"],
    ["small bedroom", "bright study", "disgusting toilet"] ]

while not finished:
    print("You are in a", locations[row][col] )
    direction = input("Which way? (N/S/E/W) ")
    if direction.lower() == "n":
        if row == 0:
            print("Sorry, you can’t go that way!")
        else:
            row = row-1
    elif direction.lower() == "s":
        if row == len(locations)-1:
            print("Sorry, you can’t go that way!")
        else:
            row = row+1
    elif direction.lower() == "w":
        if col == 0:
            print("Sorry, you can’t go that way!")
        else:
            col = col-1
    elif direction.lower() == "e":
        if col == len(locations[0])-1:
            print("Sorry, you can’t go that way!")
        else:
            col = col+1
    if col == len(locations[0])-1 and row == 0:
        print("You have found a secret tunnel!")
        col=0
        row=2
    elif col == len(locations[0])-1 and row == len(locations)-1:
        print("You have found a secret tunnel!")
        col=1
        row=1
    elif col == 1 and row == 2:
        print("You have found a secret tunnel!")
        col=0
        row=0
```

You run the above program and enter the following sequence.

E *Enter* S *Enter* N *Enter* N *Enter* E *Enter* E *Enter* N *Enter*

Enter simply means you press the Enter key after typing the letter. No spaces are entered.

After you have finished entering the above sequence of letters, what is the **name of the location** that you are in?

Answer: The name of the location is: ____

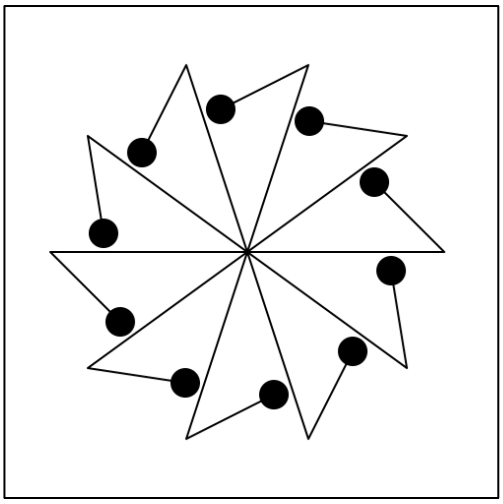
Q13) ***8 marks***

Here is a Python program.

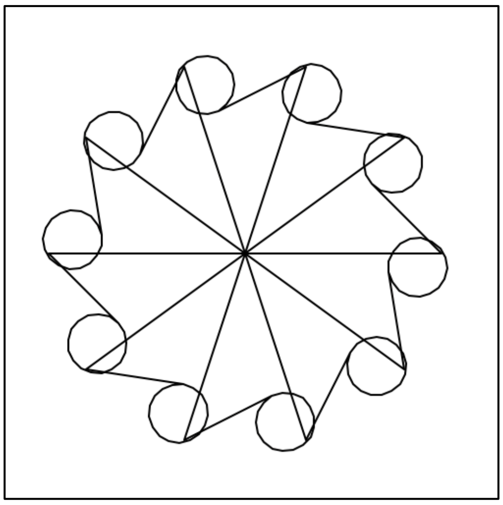
```
import turtle
instructions=["f200", "r45", "b100", "d30"]
turtle.tracer(False)
turtle.hideturtle()
turtles = []
for i in range(0, 10):
    t = turtle.Turtle()
    t.left(360 / 10 * i)
    t.width(2)
    t.hideturtle()
    turtles.append(t)
for inst in instructions:
    c = inst[0]
    n = int(inst[1:])
    for i in range(10):
        if c == "f":
            turtles[i].forward(n)
        elif c == "b":
            turtles[i].backward(n)
        elif c == "r":
            turtles[i].right(n)
        elif c == "l":
            turtles[i].left(n)
        elif c == "d":
            turtles[i].dot(n)
        elif c == "c":
            turtles[i].circle(n)
        turtle.update() # Update the display
turtle.done()
```

Which one of the following is shown after the program has finished drawing?

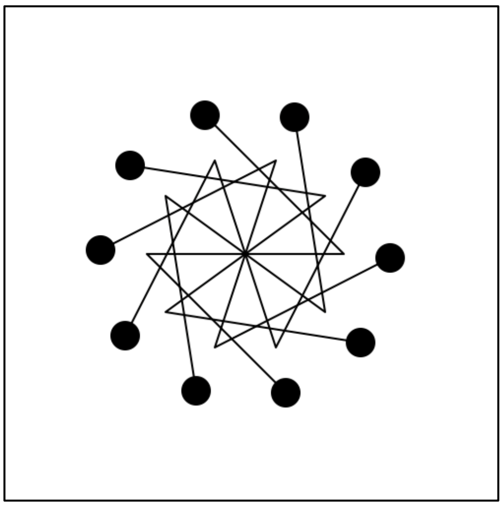
A)



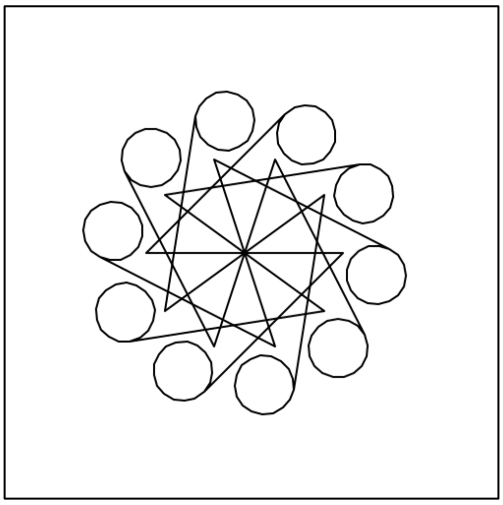
B)



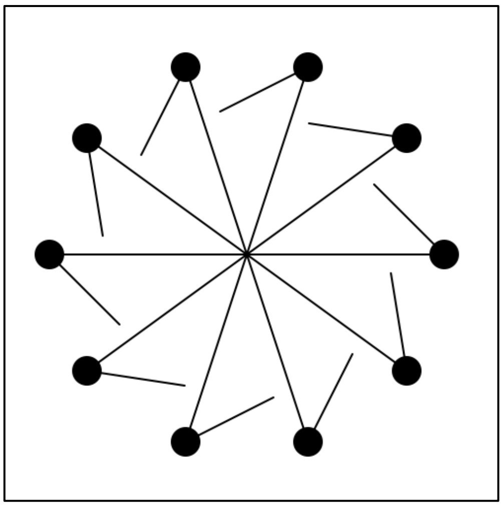
C)



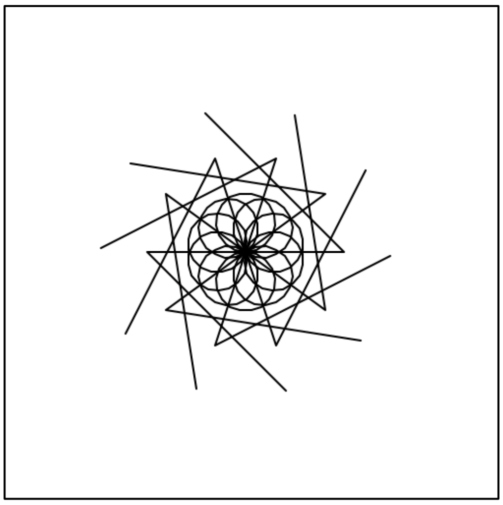
D)



E)



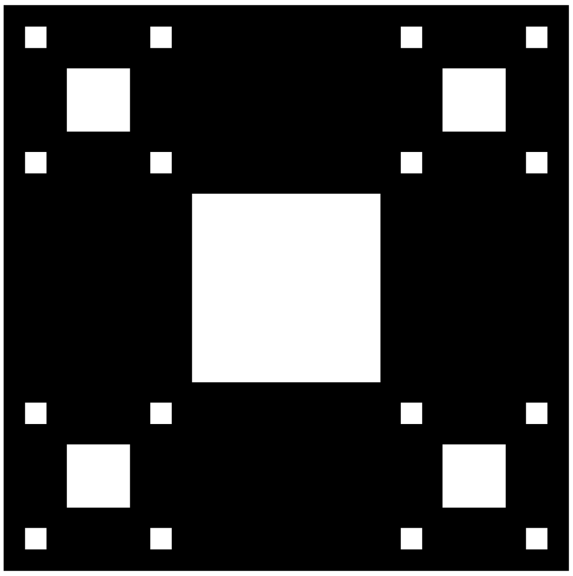
F)



Answer (A/B/C/D/E/F): ____________

Q14) ***8 marks***

The picture on the right is a black-and-white recursive pattern created by the Python program shown below.



You need to fill in the four blanks in the `carpet` function so that the program draws the picture shown on the right correctly. You cannot amend any of the code already given to you.

This question is suitable for this semester after you learn recursion

```
import turtle

max_depth = 2

def square(x, y, size, color):
    turtle.goto(x - size / 2, \
                y - size / 2)

    turtle.color(color, color)
    turtle.down()
    turtle.begin_fill()
    for _ in range(4):
        turtle.forward(size)
        turtle.left(90)
    turtle.end_fill()
    turtle.up()

def carpet(x, y, size, depth):


    square( ______________________ , ______________________ , \

            ______________________ , ______________________ )

    if depth < max_depth:
        carpet(x - size / 3, y - size / 3, size / 3, depth + 1)
        carpet(x - size / 3, y + size / 3, size / 3, depth + 1)
        carpet(x + size / 3, y - size / 3, size / 3, depth + 1)
        carpet(x + size / 3, y + size / 3, size / 3, depth + 1)

turtle.hideturtle()
turtle.speed(0)
turtle.up()

square(0, 0, 500, "black")
carpet(0, 0, 500, 0)

turtle.done()
```

Q15) ***8 marks***

A list of exams and their participating students have been stored in a text file as shown below.

```
Comp1021        Hall    20122203        20123305        20121102
Comp2011        LTA     20122203        20123305
Comp2012        LTB     20122203        20123305
Comp2711        LTJ     20122203        20110005        20131033
```

Each line in the above text file contains the course code, the exam venue and the IDs of the students taking the exam. All of them, including the student IDs, are separated by the tab character.

Based on the text file, an exam enquiry system has been created in Python. The following examples show the output of running the system using different student IDs.

Running the system by entering 20123305

```
Please enter your student ID: 20123305
You have the following exam(s):
Comp1021 (Hall)
Comp2011 (LTA)
Comp2012 (LTB)
```

Running the system by entering 20121102

```
Please enter your student ID: 20121102
You have the following exam(s):
Comp1021 (Hall)
```

Running the system by entering 20121105

```
Please enter your student ID: 20121105
You do not have exams!
```

The code of the system is shown on the next page.

You need to fill in the four blanks in the code so that the system works as shown in the above examples. You cannot amend any of the code already given to you.

```
exams = []

myfile = open("exams.txt", "r")
for line in myfile:
    data = line.rstrip().split("\t")


    exams.append({ "code" :     ________________________________ , \

                   "venue" :    ________________________________ , \

                   "students" : ________________________________ })
myfile.close()

student_id = input("Please enter your student ID: ")

result = ""
for exam in exams:
    if student_id in exam["students"]:
        result += exam["code"] + " (" + exam["venue"] + ")" +

                   ____________________________________________

if result:
    print("You have the following exam(s):")
    print(result.rstrip())
else:
    print("You do not have exams!")
```

Q16) ***10 marks***

The following code creates a dictionary called `pokemon_list`.

```
pokemon_list = {
    "Aerodactyl" :
        {"Category" : "Very Rare",
         "Level" : 8, "Position" : (100, 45)},
    "Arbok" :
        {"Category" : "Rare",
         "Level" : 1, "Position" : (-4, 105)},
    "Butterfree" :
        {"Category" : "Very Rare",
         "Level" : 8, "Position" : (30, -170)},
    "Caterpie" :
        {"Category" : "Common",
         "Level" : 5, "Position" : (-100, 120)},
    "Doduo" :
        {"Category" : "Common",
         "Level" : 3, "Position" : (190, 225)},
    "Gloom" :
        {"Category" : "Very Rare",
         "Level" : 2, "Position" : (-232, -50)},
    "Golem" :
        {"Category" : "Super Rare",
         "Level" : 2, "Position" : (-120, -145)},
    "Horsea" :
        {"Category" : "Rare",
         "Level" : 4, "Position" : (125, 20)},
    "Mankey" :
        {"Category" : "Common",
         "Level" : 5, "Position" : (-242, -35)},
    "Pikachu" :
        {"Category" : "Rare",
         "Level" : 8, "Position" : (230, 130)},
    "Raichu" :
        {"Category" : "Super Rare",
         "Level" : 10, "Position" : (-150, 159)},
    "Weedle" :
        {"Category" : "Common",
         "Level" : 8, "Position" : (-15, -170)}
}
```

Using the dictionary shown above, the Python code on the next page puts the Pokémon, as coloured circles, on a map. The user can then click anywhere on one of the circles to show the name of the corresponding Pokémon at the centre of the map.

You need to fill in the five blanks in the code so that the code works as described above. You cannot amend any of the code already given to you.

```
import turtle

pokemon_colors = {
    "Common" : "green", "Rare" : "yellow",
    "Very Rare" : "orange", "Super Rare" : "red",
}

turtle.setup(500, 500)
turtle.bgpic("map.gif")
turtle.up()
turtle.hideturtle()
turtle.tracer(False)
```

You haven't learned .ontimer(), .update() and .distance() in this semester and won't be able to answer this question.

```
def hide_text():
    text_turtle.clear()
    turtle.update() # Update the display

def show_text(x, y):
    for pokemon in pokemon_list:
        pos = pokemon_list[pokemon]["Position"]
        level = pokemon_list[pokemon]["Level"]


        text_turtle.goto( ______________ , ______________ )


        if text_turtle.distance( ______________ , ______________ ) < \

            ______________________________________________________ :
            text_turtle.home()
            text_turtle.write(pokemon, align="center", \
                              font=("Arial", 14, "bold"))
            turtle.update() # Update the display
            turtle.ontimer(hide_text, 500)

for data in pokemon_list.values():
    turtle.color(pokemon_colors[data["Category"]])
    turtle.goto(data["Position"][0], data["Position"][1])
    turtle.dot(2 * data["Level"] + 10)

text_turtle = turtle.Turtle()
text_turtle.up()
text_turtle.hideturtle()
text_turtle.color("white")

turtle.onscreenclick(show_text)

turtle.update() # Update the display
turtle.done()
```

Q17) ***14 marks***

In your music labs, you worked with a piece of music stored in a single list of music events.

In this question, a piece of music can contain more than one list of music events. The complete piece of music is stored in a dictionary. Each item in the dictionary is a list of music events associated by the name of the list. All the events in the lists are played according to their time values. Here is an example music with two lists of music events.

The music system explored in Spring 2017 is different to the music system you have experienced. It may be hard for you to answer this question because the music list structure and the method for generating the music is different.

```
# Music with multiple lists of events
music = { "treble" :
           [[0,    67, "on" ],
            [0,    76, "on" ],
            [0.25, 67, "off"],
            [0.25, 76, "off"],
            [0.25, 74, "on" ],
            [0.5,  74, "off"],
            [0.5,  65, "on" ],
            [0.5,  72, "on" ],
            [0.75, 65, "off"],
            [0.75, 72, "off"],
            [0.75, 74, "on" ],
            [1,    74, "off"],
            [1,    67, "on" ],
            [1,    76, "on" ],
            [1.25, 67, "off"],
            [1.25, 76, "off"],
            [1.25, 76, "on" ],
            [1.5,  76, "off"],
            [1.5,  67, "on" ],
            [1.5,  76, "on" ],
            [2,    67, "off"],
            [2,    76, "off"]],
           "bass" :
           [[0,    48, "on" ],
            [0,    59, "on" ],
            [0.5,  48, "off"],
            [0.5,  59, "off"],
            [0.5,  50, "on" ],
            [0.5,  60, "on" ],
            [1,    50, "off"],
            [1,    60, "off"],
            [1,    52, "on" ],
            [1,    62, "on" ],
            [1.5,  52, "off"],
            [1.5,  62, "off"],
            [1.5,  52, "on" ],
            [1.5,  60, "on" ],
            [2,    52, "off"],
            [2,    60, "off"]] }
```

Part A.

The lists in the music are played **at the same time**. The following Python code has been created to do that. Before running the code, the PyGame module, i.e. `pygame.midi`, and the time module have been imported and initialized correctly.

You need to fill in the five blanks in the code so that it can play a complete piece of music with multiple lists of events. The comments (shown below in bold) can help you understand what the code is doing. You cannot amend any of the code already given to you.

```
# The indices dictionary stores, for each event list, the index of
# the current music event that the code is looking at
indices = {}

# Initially, the code starts from the first music event for each list
for key in music:


    indices[ _______________ ] = _______________

# The current playback time of the music that the code is working on
current_time = 0

finished = False
while not finished:
    # For each event list, process the music event that occurs
    # at the current playback time
    for key in music:
        index = indices[key]

        # All events for this list have been completed
        if index >= len(music[key]):
            continue

        # If the current event of the list occurs at the current time
        if music[key][index][0] == current_time:
            event = music[key][index]

            # Run the code for the note on or off of the event
            if event[2] == "on":
                output.note_on(event[1], 127)
            else:
                output.note_off(event[1], 127)

            # Move on to the next event of this event list


            indices[key] = __________________________________________
```

Continued on the next page…

Continued (with the indentation indicated by the vertical dashed line)…

```
# Find the playback time of the next upcoming event,
# i.e. the one closest to the current playback time
next_event_time = -1

# Find the earliest event among all the upcoming events
for key in music:
    index = indices[key]

    # All events for this list have been completed
    if index >= len(music[key]):
        continue

    # Update the next event time to be the earliest one so far
    if next_event_time == -1 or \


       next_event_time > ________________________________________ :


        next_event_time = ________________________________________

# Stop the playback if there are no more events left
if next_event_time == -1:
    finished = True

# Otherwise, sleep and move on to the next event time
else:
    time.sleep(next_event_time - current_time)
    current_time = next_event_time
```

Part B.

Let's assume that the code shown in Part A has been completed and is working correctly. Here is another piece of music with multiple lists of events.

```
# Do-re-mi-fa-so
music = { "do" : [[1, 60, "on"], [2, 60, "off"]],
          "re" : [[3, 62, "on"], [4, 62, "off"]],
          "mi" : [[5, 64, "on"], [6, 64, "off"]],
          "fa" : [[2, 65, "on"], [3, 65, "off"]],
          "so" : [[4, 67, "on"], [5, 67, "off"]] }
```

How many music notes do you hear in total if you play the above music using the code in Part A?

Answer (0/1/2/3/4/5): __________

- End of the exam -