1 (20)	2 (20)	3 (20)	4 (20)	5 (20)	TOTAL (100)

[CS101] Introduction to Programming 2015 Spring - Midterm Examination

SECTION	STUDENT ID	NAME

※ Please check if you received all 22 pages of the test material.
※ 시작하기 전에 반드시 페이지의 수를 확인 하십시오. (전체: 22쪽)

※ Fill in your student identification number and name. Otherwise you will lose 1 point for each missing piece of information.
※ 위의 정보(학번,이름)를 정확히 기입하지 않을 경우, 각 실수 당 1점이 감점 됩니다.

※ **TAs will not answer your questions about the exam**. If you think that there is anything ambiguous, unclear or wrong about a problem, please write the reasons and make necessary assumptions to solve the problem. We will take your explanation into consideration while grading.
※ **시험시간동안 질문을 받지 않습니다**. 만일 문제에 오류나 문제가 있을 경우, 왜 문제가 이상이 있다고 생각하는지에 대해서 기술하시면 되겠습니다. 또한 문제가 애매하다고 생각되는 경우 문제를 푸실 때 본인이 생각하는 가정을 함께 작성하셔서 문제를 푸시면 되겠습니다. 채점 시 가정 및 설명을 고려하도록 하겠습니다.

1. (20 points) Answer each question according to the instructions.

1-1. (1 point) Two object types, 'list' and 'tuple', are used to abstract sequences. Which is mutable?

____

1-2. (1 point) Yes or No?

'string' is mutable.

____

1-3. (4 points) What is the result of the following program?
(**※ Note: Assume that Python version number is 2.x.x**)

```
print 10 / 5

print 13.0 // 4.0

print 1 + 3.0 * 2 ** 4 / 4

print "Hello" + "CS101" * 2
```

(1-3)	

1-4. (4 points) When the following four statements are executed, they will output an error. Match the statements with the types of error listed below.
(**※ Note: Statements (1) to (4) and errors (a) to (d) are 1:1 mapped.**)

①	②

`tuple = ("CS101", "A+", "Happy")`
`tuple[1] = "C-"`
	`print 10 / (2-2)`

③	④

`from math import sin, pi`
`print sin(pi/4)`
`print cos(pi/4)`
	`while True print 'Hello world'`

(a) SyntaxError	**(b)** TypeError	**(c)** ZeroDivisionError	**(d)** NameError

(1-4)	①	②
	③	④

1-5. (4 points) The following programs show different results. Please write down the result of each program.

①	②	③

```
a = 17

def test():
    a = 13
    print a

test()
```
	```
	a = 17

	def test():
	    print a

	test()
	```
		```
		a = 17

		def test(a):
		    print a

		test(23)
		```

(1-5)	①	②
	③	

1-6. (6 points) True or false?
(**※ Note: 1 point for each question.**)

Statement	**Answer (T/F)**
(**※ Example**) Variables defined outside of a function are called **global variables**.	**T**
Variable names in Python are **case sensitive**. For example, **Pi** and **pi** refer to the different variable.	
An **object** can be accessed with **multiple variables**. They are called aliases.	
A '**for**' or '**while**' loop cannot run infinitely. Infinite loops will be detected as errors before the program executes the loop statements.	
Python always makes sure to evaluate **every operand** in the boolean expression, even though some operands do not need to be computed. For example, in the boolean expression '(3*2==6) or (3+3==3)', Python always evaluate the both boolean terms (3*2==6) and (3+3==3), even the whole expression become True regardless of the result of the second term, (3+3==3).	
A **function** can be an argument of another function.	
A **function** may return nothing. In other words, we can define a function without using a '**return**' statement.	

2. (20 points) Answer each question according to the following instruction.

2-1. (3 points) Make a conditional statement which has the same result with the following conditional statement **without using the keyword 'or'**.
(**※ Note: If you use the keyword 'or', you will get 0 point.**)
(**※ Hint: You may use '(', ')' and the keywords 'not' and 'and'.**)

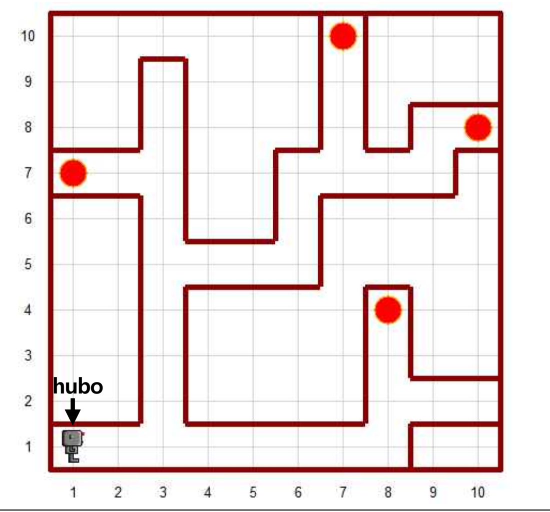
```
not hubo.on_beeper() or hubo.left_is_clear() or hubo.right_is_clear()
```

(2-1)	

2-2. (4 points) The following program is one of the solutions for the programming homework #1. How many times will ① and ② be executed while the program is running for the given world?

```
hubo.drop_beeper()
hubo.move()
while not hubo.on_beeper():
    if hubo.left_is_clear():
        hubo.turn_left() // ①
    if hubo.front_is_clear():
        hubo.move()
    else:
        if hubo.on_bomb(): // ②
            hubo.defuse_bomb()
        for i in range(3):
            hubo.turn_left()
```

Initial world



(2-2)	①	②

2-3. (13 points) You are asked to implement four functions, "generate_food", "is_game_over", "is_this_cell_snakes_neck" and "change_direction" for "Snake 101" that you worked on for the programming homework #2.

The followings are the descriptions for the global variables used in those functions.

Name	**Type**	**Description**
BOARD_SIZE_X	int	the size of the board in the number of cells for x-axis.
BOARD_SIZE_Y	int	the size of the board in the number of cells for y-axis.
snake_head_x	int	the location of the snake's head in x-axis.
snake_head_y	int	the location of the snake's head in y-axis.
snake_direction	str	the direction of the snake that is one of 'N', 'S', 'W' and 'E' which represent north, south, west and east, respectively.
snake_trail	list	the trail of the snake (from tail to head) storing the history of the recent locations for the snake's head. **※ Note: Each element is a tuple of two integers.** **(i.e.) snake_trail[-1] == (snake_head_x, snake_head_y)**
food_position_x	int	the location of the food in x-axis.
food_position_y	int	the location of the food in y-axis.

2-3-1. (3 points) Fill in the blank to complete the 'generate_food' function.

Function signature	generate_food()
Return value	None
Description	This function is called to create a food object that a snake can eat. **※ Note: You need to avoid the cells where the snake resides (snake_trail) when generating the food.**

```
def generate_food():
    global food_position_x, food_position_y
    food_position_x, food_position_y = snake_trail[-1]
    while [            (2-3-1)            ] :
        food_position_x = random.randint(0, BOARD_SIZE_X - 1)
        food_position_y = random.randint(0, BOARD_SIZE_Y - 1)
```

(2-3-1)	

2-3-2. (4 points) Fill in the blank to complete the 'is_game_over' function.

Function signature	is_game_over()
Return value	**True** if the game over condition is met; **False** otherwise.
Description	This function is called to check if a game is over. **※ Note: There are two possible conditions when the game is over.** **(1) The snake collided itself.** **(2) The snake collided to the wall.** **(i.e.) One of snake_head_x and snake_head_y is out of bound.**

```
def is_game_over():
    (2-3-2)
```

(2-3-2)	

2-3-3. (2 points) Fill in the blank to complete the 'is_this_cell_snakes_neck' function.

Function signature	is_this_cell_snakes_neck(x, y)
Parameter 1	x (int): x-axis cell index to query
Parameter 2	y (int): y-axis cell index to query
Return value	**True** if the given cell location is the neck; **False** otherwise.
Description	This function is called to check if the given cell location is the former position of the snake's head. **(i.e.) the snake's neck**

```
def is_this_cell_snakes_neck(x, y):
   return len(snake_trail) > 1 and [        (2-3-3)        ]
```

(2-3-3)	

2-3-4. (4 points) Fill in the blank to complete the 'change_direction' function. The function must call the 'is_this_cell_snakes_neck' function and it is assumed that the 'is_this_cell_snakes_neck' function has been correctly implemented.

Function signature	change_direction(new_direction)
Parameter 1	new_direction (str): the direction a user intended to change **(i.e.) one of 'N', 'S', 'W' and 'E'**
Return value	None
Description	This function is called if a player intends to change the direction. Change 'snake_direction' to 'new_direction' if it is a valid turn. **※ Note: It is invalid to change the direction toward the cell which was the snake's last head position.** **(i.e.) the snake's neck**

```
def change_direction(new_direction):
  global snake_direction
  [                    (2-3-4)                    ]
```

(2-3-4)	

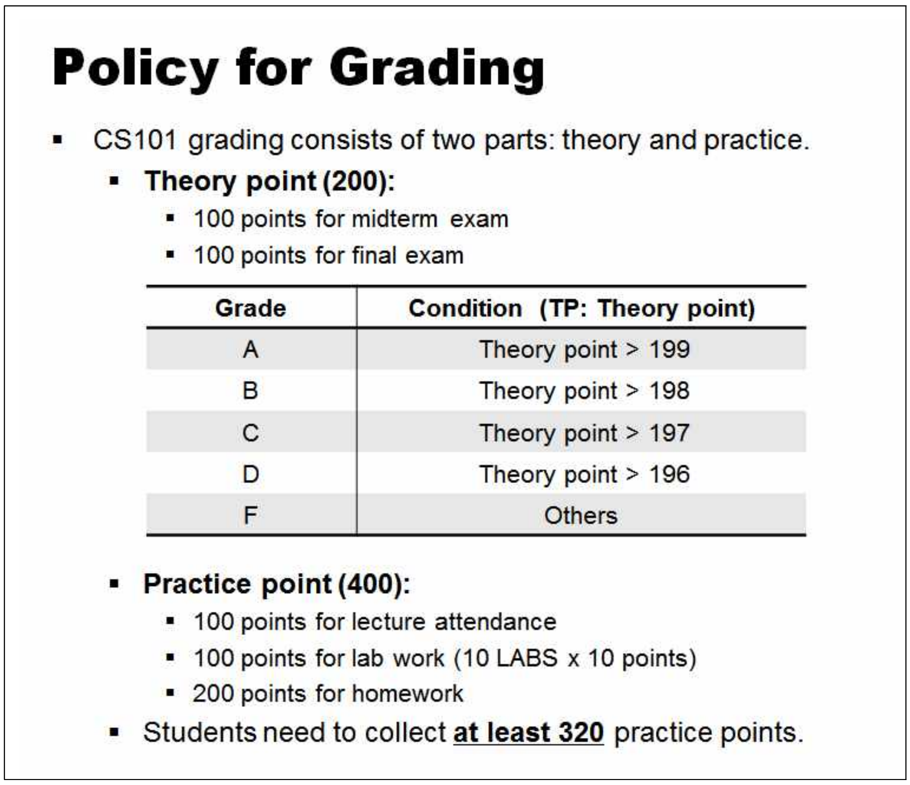
3. (20 points) Answer each question according to the instruction.

3-1. (4 points) What is the result of the following program?

```
def fib(n):
  a, b = 0, 1
  while a < n:
      print a,
      a, b = b, a+b
fib(100)
```

(3-1)	

3-2. (4 points) You are asked to make a program which follows the policy for grading in CS101 course. Your program will be used for assigning and printing out the grade. F grades are given to those whose practice points are under 320 or theory points are under 196. Your program must show the same results as the example below.



Policy for Grading

- CS101 grading consists of two parts: theory and practice.
  - **Theory point (200):**
    - 100 points for midterm exam
    - 100 points for final exam

Grade	Condition (TP: Theory point)
A	Theory point > 199
B	Theory point > 198
C	Theory point > 197
D	Theory point > 196
F	Others

  - **Practice point (400):**
    - 100 points for lecture attendance
    - 100 points for lab work (10 LABS x 10 points)
    - 200 points for homework
  - Students need to collect **at least 320** practice points.

```
while True :
    p_point = int(raw_input('What is your practice point: '))
    t_point = int(raw_input('What is your theory  point: '))
```

(3-2)

```
    print 'Your grade : ' + grade
```

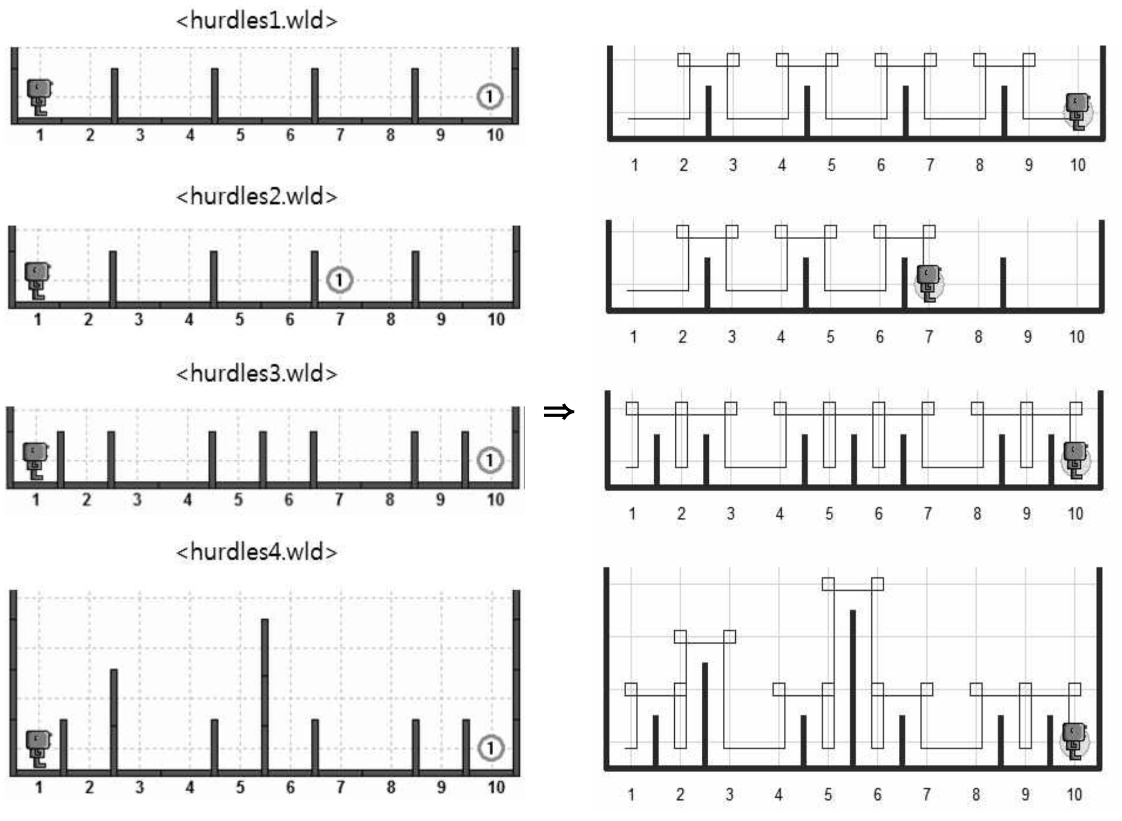
Example of the program

```
>>>What is your practice point: 300
>>>What is your theory  point: 200
Your grade : F
>>>What is your practice point: 350
>>>What is your theory  point: 199
Your grade : B
```

(3-2)	

3-3. (6 points) You are asked to make a program that will complete tasks in four worlds which are from hurdles1 to hurdles4. When the program start, a robot is put on (1, 1) in each world facing the east. And the program ends when the robot is on the beeper. Fill in the blank to make the following program work with four worlds.

<hurdles1~4.wld>



```
from cs1robots import *

load_world("worlds/hurdles1.wld")
#load_world("worlds/hurdles2.wld")
#load_world("worlds/hurdles3.wld")
#load_world("worlds/hurdles4.wld")

hubo = Robot()
hubo.set_trace("blue")

def turn_right():
```

(3-3-1)

```
def move_jump_or_finish():
    # test for end of race
    if not hubo.on_beeper():
        # is there a hurdle?
        if hubo.front_is_clear():
            hubo.move()
        else:
            jump_one_hurdle()

def jump_one_hurdle():
    ┌──────────────────────────────────────┐
    │               (3-3-2)                │
    └──────────────────────────────────────┘

for i in range(20):
    move_jump_or_finish()
```

(3-3-1)	

(3-3-2)	

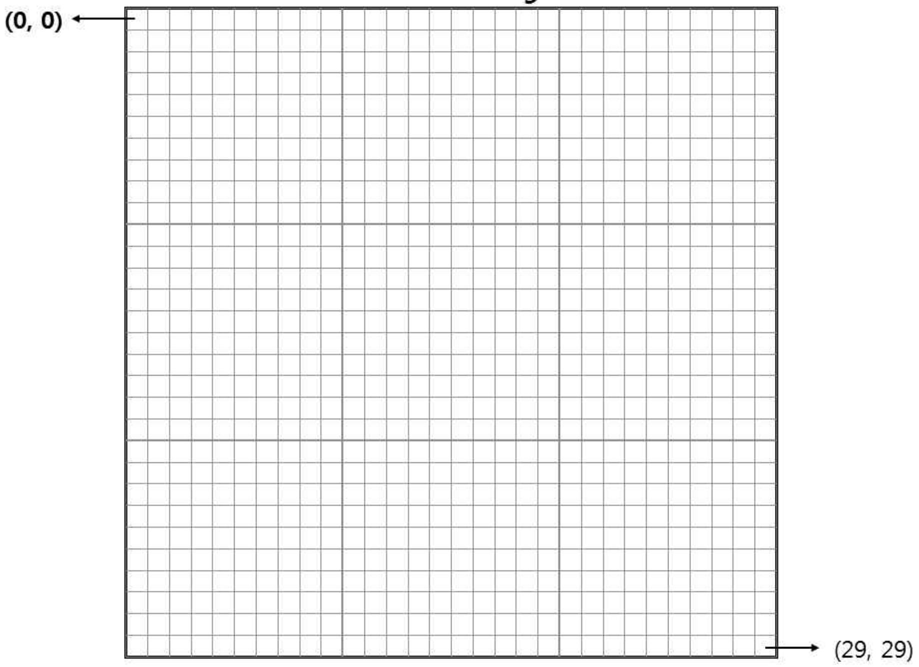
3-4. (6 points) Draw the result of the following program on the following canvas.

```
from cs1media import*
from math import*

black = (0,0,0)
white = (255,255,255)

size = 30
img = create_picture(size,size,white)

for i in range(size):
    for j in range(size):
        if sin(2*pi/size*(i+1)) < 0:
            if sin(2*pi/size*(j+1)) < 0:
                img.set(i,j,black)
        else:
            if sin(2*pi/size*(j+1)) >= 0:
                img.set(i,j,black)
img.show()
```

※ Hint:

```
>>>sin(pi)
1.2246467991473532e-16
>>>sin(2*pi)
-2.4492935982947064e-16
```

<Answer for *img*>



4. (20 points) Answer each question according to the instruction.

4-1. (8 points) What is the result of the following program?

```
def func1(var2, var3):
   var2 += 2
   var3 = 0

def func2(var1):
   var1 += 5
   var3 = 8
   return var1

def func3(var):
   global var2, var4
   var = var2
   var4 *= 2

func1(var2, var3)
print var1, var2, var3, var4

var1 = func2(var3)
print var1, var2, var3, var4

var2 = func3(func2(2))
print var1, var2, var3, var4
```

(4-1)	

4-2 (3 points) Please implement the following functions in Python
(**※ Note: 'function2' must call 'function1'.**)

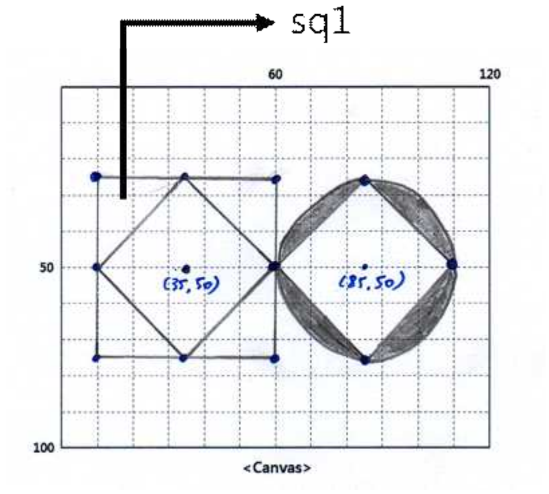
$$f_1(x) = 5x + 10$$
$$f_2(x,y) = xf_1(y) + yf_1(3x)$$

```
def function1( x ) :
        (4-2-1)

def function2( x, y ) :
        (4-2-2)
```

(4-2-1)	

(4-2-2)	

4-3. (9 points) Complete the code to draw the following painting which includes four figures(= an outer square, an outer circle, two inner diamonds) on the canvas. The source code for drawing the outer square(= sq1) is given. So, please write the code for drawing the rest of the figures. **(i.e.) an outer circle and two inner diamonds**



The followings are the descriptions for the objects and their methods for (4-3).

Function	Description
Canvas(width, height, ‘backgroundColor’, ‘title’)	Create a new canvas for drawing.
canvas.add(object)	Add a Drawable object to the canvas.
Square(length of a side, centerPoint(x, y))	Return a new Square object.
Circle(radius, centerPoint(x, y))	Return a new Circle object.
sqrt(number)	Return the square root of the number.
object.setDepth(depth)	Set the depth of the object. **※ Note: Objects with smaller depth appear in foreground.**
object.setFillColor(‘color’)	Set the interior color of the object to the given
object.rotate(angle in a degree)	Rotate the object around its center point.
object.scale(scaling factor)	Make an object smaller or larger with scaling factor
object.clone()	Return a duplicate of the drawable object.
object.move(dx, dy)	Move the object dx units along X-axis and dy units along Y-axis.

```
from cs1graphics import *
from math import *

paper = Canvas( 120, 100, 'white', 'Canvas' )

sq1 = Square( 50, Point( 35,50 ) )
sq1.setFillColor( 'white' )
sq1.setDepth( 100 )
paper.add( sq1 )
```

(4-3)

(4-3)	

5. (20 points) The following program shows the game history of top 5 clubs in the 2015 pro-baseball season until 04/01/2015. Answer each question according to the following instruction.

```
clubs = ["Doosan", "Hanhwa", "KIA", "Lotte", "Samsung"]
outcomes = ["Victory", "Defeat"]
history = [("Doosan", "Victory", 3), ("Hanhwa", "Victory", 1), ("Hanhwa",
"Defeat", 2), ("KIA", "Victory", 3), ("Lotte", "Victory", 3), ("Lotte",
"Defeat", 1), ("Samsung", "Victory", 3), ("Samsung", "Defeat", 1)]
merged_history = []

                              (5-1)

##########################
print history            # - (1) first display
##########################

def merge():
    for club in clubs:
        merged_history.append([club, 0, 0])
        for one in history:
            if one[0] == club:
                if one[1] == "Victory":
                         (5-2-1) Set the number of wins for 'club'.
                elif one[1] == "Defeat":
                         (5-2-2) Set the number of loses for 'club'.
        (5-2-3) Change the type of the element of 'merged_history'
                        from 'list' to 'tuple'.

merge()

##########################
print merged_history     # - (2) second display
##########################
```

```
def winningRate(club):
    ┌──────────────────────────────────────┐
    │                (5-3)                 │
    └──────────────────────────────────────┘

def compare(club1, club2):
    winningRate_club1 = winningRate(club1)
    winningRate_club2 = winningRate(club2)
    ┌──────────────────────────────────────┐
    │                (5-4)                 │
    └──────────────────────────────────────┘

merged_history.sort(compare)

##########################
print merged_history     # - (3) third display
##########################
```

5-1. (6 points) In 04/02/2015, "Hanhwa" wins a game against "Doosan". Fill in the blank to update the elements of 'history' list. After update, the result of printing the elements in 'history' list must be same as the following. **(i.e.) reflect the result of 04/02/2015**
(**※ Hint: The answers can be multiple lines of codes.**)

Execution result of 'print history' which is 'first display' part

```
[('Doosan', 'Victory', 3), ('Doosan', 'Defeat', 1), ('Hanhwa', 'Victory',
2), ('Hanhwa', 'Defeat', 2), ('KIA', 'Victory', 3), ('Lotte', 'Victory',
3), ('Lotte', 'Defeat', 1), ('Samsung', 'Victory', 3), ('Samsung',
'Defeat', 1)]
```

(5-1)	

5-2. (6 points) The ‘merge’ function fills ‘merged_history’ list with a set of tuples whose elements are a club name, the number of the club’s wins and the number of the club’s loses. Fill in three blanks inside the ‘merge’ function to update the elements of ‘merged_history’ list. After update, the result of printing the elements in ‘merged_history’ list must be same as the following.
(※ **Hint: The types of a club name, the number of wins and the number of loses are string, non-negative integer and non-negative integer, respectively.**)

Execution result of ‘print merged_history’ which is ‘second display’ part
`[('Doosan', 3, 1), ('Hanhwa', 2, 2), ('KIA', 3, 0), ('Lotte', 3, 1), ('Samsung', 3, 1)]`

(5-2-1)	

(5-2-2)	

(5-2-3)	

5-3. (3 points) The function ‘winningRate’ calculates and returns a winning rate of the ‘club’. Winning rate is calculated as follows:

$$\text{winning rate} = \frac{\text{the number of wins}}{\text{the number of games}}$$

Complete the ‘winningRate’ function so that it returns winning rate of the ‘club’.
(※ **Hint: The type of the return value is float.**)

Example of ‘winningRate’ function

`>>> winningRate((“My club”, 3, 5))`
`0.375`

(5-3)	

5-4. (5 points) The ‘compare’ function is given as an argument to ‘sort’ method of a list object and the ‘compare’ function is used while sorting the elements of the list object. Complete the ‘compare’ function such that ‘merged_history’ list is sorted in descending order of club’s winning rates.
(**※ Note 1: If there are elements whose winning rates are same, then they must be sorted in ascending order of club’s names.**
※ Note 2: This function should work for general unsorted input list.)

Execution result of ‘print merged_history’ which is ‘third display’ part
`[('KIA', 3, 0), ('Doosan', 3, 1), ('Lotte', 3, 1), ('Samsung', 3, 1), ('Hanhwa', 2, 2)]`

(5-4)	