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

[CS101] Introduction to Programming
2016 Fall - Midterm Examination

SECTION	STUDENT ID	NAME

※ Please check if you have all 23 pages of the test material.
※ 시작하기 전에 반드시 페이지의 수를 확인 하십시오.(전체 : 23쪽)

※ 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 there is anything ambiguous, unclear or wrong about a problem, please write down 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.

※ **Stick to Python 3**. We will grade answers only in Python 3.
※ **파이썬 3만 사용하십시오**. 채점은 파이썬 3 기준으로만 합니다.

1-1. (8 points/2 point each) For each pair of code (a) and (b), write 'SAME' if they produce the exact same output. If they do not produce the same output, make one small and simple change to (a) such that they produce the same code.

(a)		(b)
`print(type("hello"))`		`print(type('hello'))`
Answer		

(a)		(b)
`print(13.4//5)`		`print(12//6)`
Answer		

(a)		(b)

`print(type(12/5))`		`from math import pi`
		`print(type(pi))`

Answer		

(a)		(b)

`i = 5`
`while (i < 10):`
`    print(i)`
`    i = i + 1`
		`for j in range(5,10):`
		`    j = j + 1`
		`    print(j)`

Answer		

1-2. (2 points) Consider the for loop below. Write a while loop that produces the same output as this for loop.

Statement	Answer

`for i in range(0,20,4):`
`    print(i)`	

1-3. (10 points / 2 points each) What is the output of the following code?

Statement	Answer

`a = [1, 2, 3]`
`b = a`
`b[1] = 5`
`print(a)`	

`a = (10, 20, 30)`
`b = a`
`for i in range(0,3):`
    `b[i] = b[i] + 5`
    `print(a)`	

`a = [10, 20, 30]`
`b = [10, 20, 30]`
`b[2] = 50`
`print(a)`	

`x = [1,4,7,2,9,5]`
`x.sort()`
`x.reverse()`
`y = x[:-3]`
`a,b,c = y`
`print(c)`	

`x = [1,2]`
`y = [3,4]`
`x.append(y)`
`a,b,c = x`
`print(c)`	

2. (20 points) Please read the instructions carefully and answer according to the instruction. For all programming answers, make sure you use clear indentation.

2-1. (4 points) Fill in the blank parts in the [program code 2-1] to complete the four functions: `turn_right()`, `turn_around()`, `face_north()` and `pickup_items()`

You may use the following functions in cs1robots module that you're already familiar with: `set_trace()`, `set_pause()`, `get_pos()`, `move()`, `turn_left()`, `front_is_clear()`, `left_is_clear()`, `right_is_clear()`, `facing_north()`, `on_beeper()`, `pick_beeper()`, `carries_beepers()`, `drop_beeper()`.

[Program Code 2-1]

```
from cs1robots import *
create_world()
hubo = Robot()

#Get hubo to turn right
def turn_right():
                              (2-1-1)

#Get hubo to turn around and face the opposite side
def turn_around():
                              (2-1-2)

#Get hubo to face north
def face_north():
                              (2-1-3)

#Get Hubo to pick up all items of the current position
def pickup_items():
                              (2-1-4)
```

Your Answers:

2-1-1

2-1-2

2-1-3

2-1-4

2-2 (6 points) Complete the function go_to in [program code 2-2] in which hubo travels to the position specified by the parameters avenue(column) and street(row).
Hint. You may use the functions you've defined above in question 2-1.

[Program Code 2-2]

```
def go_to(avenue, street):
    x, y = hubo.get_pos()
    if x < avenue:
        
                    2-2-1
        
        while x != avenue:
            hubo.move()
            x, y = hubo.get_pos()
    else:
        
                    2-2-2
        
        while x != avenue:
            hubo.move()
            x, y = hubo.get_pos()
    if y > street:
        
                    2-2-3
        
        while y != street:
            hubo.move()
            x, y = hubo.get_pos()
    else:
        face_north()
        while y != street:
            hubo.move()
            x, y = hubo.get_pos()
```

Your Answers:

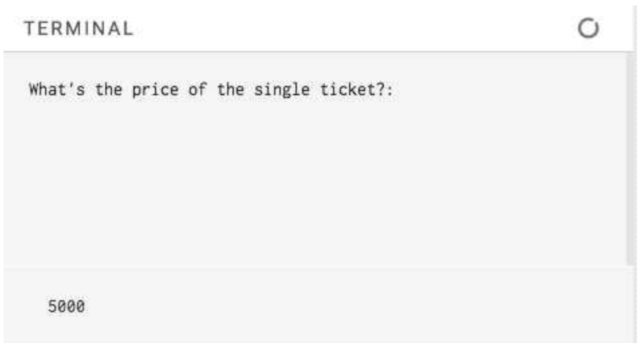
2-2-1

2-2-2

2-2-3

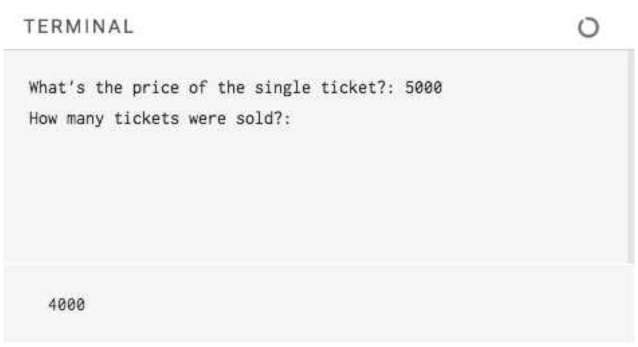
2-3 (10 points) You have a lottery program to control the prize money for each of the winning number matches (you have done this in your homework 2 when writing `def_second_policy` function). In your homework, you specified the winning prize money for the first place (ie. 100,000), and used a geometric sequence to calculate the winning prize money for the rest of the winners (ie. 50,000 for 2nd place, 25,000 for 3rd place, etc).
The usual method for doing this in the real world is to calculate the total pot size based on the number of lottery ticket sales, and share the total pot among the winners. So let's change the program to be more realistic.

2-3-1 (4 points) Write a program that receives user input for the number of ticket sales and the price of a single ticket and returns the total pot size. Define two functions, `main()` for handling user interactions and `total_sales()` for handling the actual calculation. Be careful with type conversions. An example of the user interaction is demonstrated below:


TERMINAL

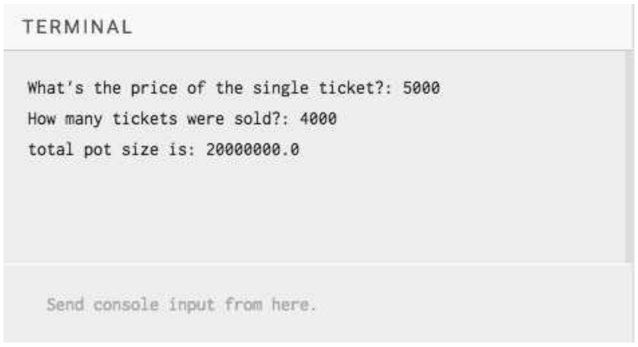
```
What's the price of the single ticket?:

  5000
```


TERMINAL

```
What's the price of the single ticket?: 5000
How many tickets were sold?:

  4000
```


TERMINAL

```
What's the price of the single ticket?: 5000
How many tickets were sold?: 4000
total pot size is: 20000000.0

  Send console input from here.
```

Your Answer:

```
# num_of_sales: integer value for the total number of ticket sales
# single_price: float value for the price of a single lottery ticket
def total_sales(num_of_sales, single_price):
```

```
def main():
```

2-3-2 (6 points) Now that we have collected the total lottery pot size in the problem above, let's write a function that distributes the total pot size. In the [Program Code 2-3], the prize money for the winners are calculated through an equation:

prize money for ith place = (1st place prize money) / 2 ** i

The sample output is demonstrated in `[ Program Output 2-3 ]`.

[Program Code 2-3]

```
#rank: integer value for the ranks
#max_number_of_correct_picks: integer value for maximum correct digits
#prize_money: float value for total lottery pot
def calc_prize_money(rank, max_number_of_correct_picks, prize_money):
    second_policy = []
    for i in range(rank):
        account = int(prize_money / (2 ** i))
        item = (i + 1, max_number_of_correct_picks - i, account)
        second_policy.append(item)
    return second_policy
```

[Program Output 2-3]

```
>> print(calc_prize_money(5,5,100000))
[(1, 5, 100000),(2, 4, 50000),(3, 3, 25000),(4, 2, 12500),(5, 1, 6250)]
```

Your job is to modify the `calc_prize_money` function (or rewrite if you want to) so that the first prize gets 1/2 of the total pot, second prize gets the 1/4 of the total pot, third prize gets the 1/8 of the total pot and the fourth prize gets 1/16 of the total pot, etc. You may assume you only print upto first 5 places. Your new function should give the following output:

[New Program Output 2-3]

```
>> print(cal_prize_money2(5,5,100000))
[(1, 5, 50000),(2, 4, 12500),(3, 3, 4687),(4, 2, 2050),(5, 1, 961)]
```

Your answer:

```
#rank: integer value for the ranks
#max_number_of_correct_picks: integer value for maximum correct digits
#prize_money: float value for total lottery pot
def calc_prize_money2(rank, max_number_of_correct_picks, prize_money):
```

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

3-1-1. (2 points) What is the result of the following program?

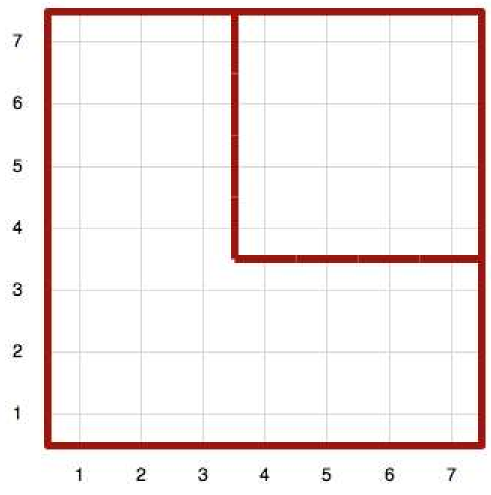
3-1-1	terminal – 2points
```
# parameter x is always assigned a real number
# (x: float or int)
def buggy_absolute(x):
    if x < 0:
        return -x
    if x > 0:
        return x
for i in range(3):
    print(buggy_absolute(-i))
```

3-1-2. (3 points) The "buggy_absolute" function have a bug and you need to debug it. Define a new absolute function and troubleshoot this problem.

3-1-2 (3points)	`def absolute(x):`

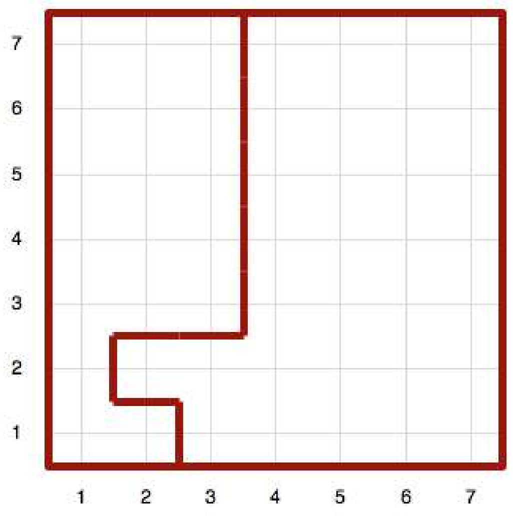
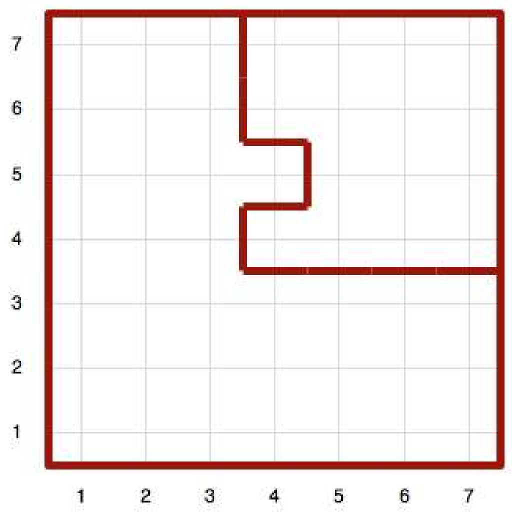
3-2. (11 points) What is the result of the following program? Draw the hubo's trace line. Hubo will start at 1street and 1avenue in all amazing world and you don't need to drawing Hubo.

python code (3-2-1)	3-2-1 (Screen) - 3points

```
from cs1robots import *
load_world('worlds/amazing2.wld')
hubo = Robot(beepers=1)
hubo.set_trace('red')
hubo.drop_beeper()
hubo.move()
while not hubo.on_beeper():
    if hubo.front_is_clear():
        hubo.move()
    else:
        hubo.turn_left()
hubo.pick_beeper()
```
	 amazing2.wld

python code (3-2- *2~3*)
(code below)

```
from cs1robots import *
load_world('worlds/amazing#.wld')    # amazing5.wld or amazing6.wld
hubo = Robot(beepers=1)
hubo.set_trace('red')
hubo.drop_beeper()
while not hubo.front_is_clear():
    hubo.turn_left()
hubo.move()
while not hubo.on_beeper():
    if hubo.right_is_clear():
        hubo.turn_left()
        hubo.turn_left()
        hubo.turn_left()
        hubo.move()
    if hubo.front_is_clear():
        hubo.move()
    else:
        hubo.turn_left()
hubo.pick_beeper()
```

3-2-2 (Screen) - 4points	3-2-3 (Screen) - 4points
 amazing5.wld	 amazing6.wld

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

```
mysay = "Niconiconi"

def myfun():
    mysay = mysay + "to die"
    return True

if (3<3 and 2>1) or 5!=6:
    print("Hello, hubo!!")
if (False or True) or myfun():
    mysay = "a nice day"
else:
    mysay = mysay+"to live"

print(mysay)
```

4-1. (10 points) Implement a program which prints the properties of a number whether the number is 1) positive or negative, 2) even or odd, 3) 0. You need to implement four functions to complete this program. The 'is_zero', 'is_positive' and 'is_even' functions return 'True' when a number is 0 or positive or even, respectively. The 'print_properties' function prints out the properties of a parameter x by calling the 'is_zero', 'is_positive' and 'is_even' functions.

※ All codes of printing messages must be implemented in the `'print_properties'` function.
※ `'print_properties'` takes only an integer input. Hence, you must check the type of x in the `'print_properties'` function.
※ In this program, 0 is not regarded as an even nor odd number. In addition, 0 is neither a positive nor negative number. Hence, if a number is 0 then you must not call the `'is_positive'` and `'is_even'` functions.
※ Please consider the output example below to implement the functions.

```
def is_zero(x):
    # 4-1-1

def is_even(x):
    # 4-1-2

def is_positive(x):
    # 4-1-3

def print_properties(x) :
    # 4-1-4
```

Output example:

```
print_properties('abc1')
>>> 'abc1' is not an integer number.
print_properties(1.1)
>>> 1.1 is not an integer number.
print_properties(0)
>>> 0 is 0, that's it.
print_properties(2)
>>> 2 is a positive and even number.
print_properties(3)
>>> 3 is a positive and odd number.
print_properties(-3)
>>> -3 is a negative and odd number.
```

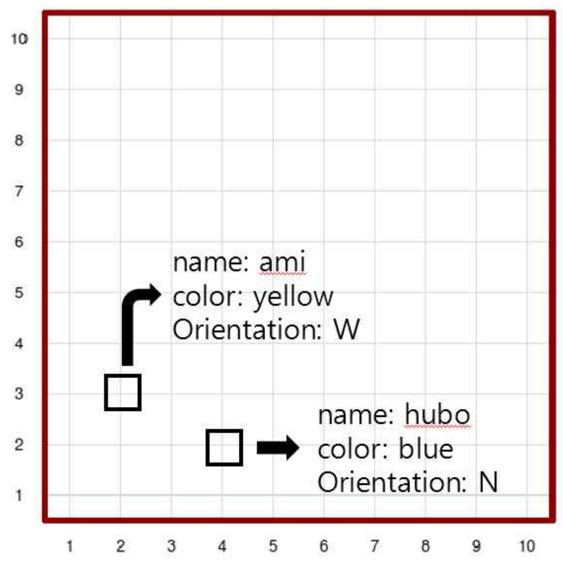
4-1-1 (1 point)	
4-1-2 (1 point)	
4-1-3 (1 point)	
4-1-4 (7 points)	

4-2. (4 points) What is the result of the following program? Draw robot objects with any shape (e.g., square) on the 10x10 canvas, but you have to explicit the **position**, **name**, **color** and **orientation**(N: North, S: South, E: East, W: West) of each robot object.

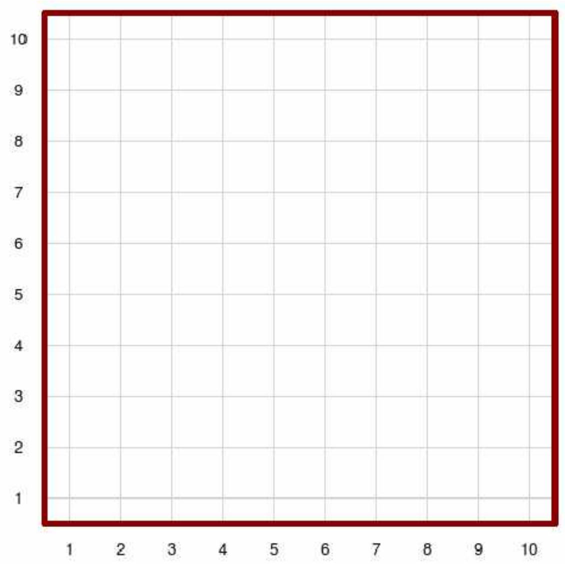
```
from cs1robots import *
create_world()

hubo = Robot("yellow")
hubo.move()
ami = hubo
ami.turn_left()
hubo.move()
hubo = Robot("blue")
hubo.move()
```

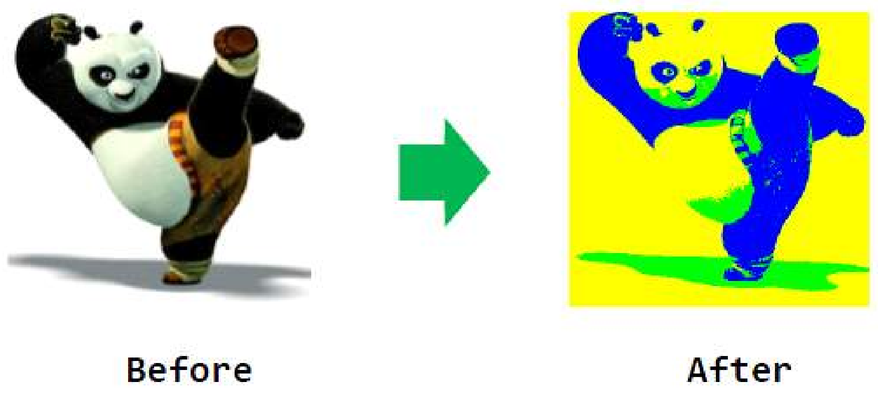
<example solution>



<4-2>



4-3. (6 points) Write a program that takes a photo and converts it into a three color poster where the bright regions are filled with yellow, the dark regions are filled with blue, and the others are filled with green. The result of this program is as follows:



Here are the descriptions of the method related to (4-2).

Function	Description
`load_picture(filename)`	Create an image by loading file called filename
`object.size()`	Return the size of the object(image) as a tuple (width, height)
`object.show()`	Display the object(image)
`object.get(x, y)`	Return pixel at x, y
`object.set(x, y, color)`	Set pixel at x, y to a color tuple

```
from cs1media import *

# Get a average luminance value from a color tuple
def luminance(color) :

            # 4-3-1


# Set thresholds
threshold1 = 255 / 3
threshold2 = (255 / 3) * 2

# Set color tuples
yellow = ( 255, 255, 0 )
green = ( 0, 255, 0 )
blue = ( 0, 0, 255 )

# Load an image
img = load_picture( "fender.jpg" )

# Convert a load image to a three color poster
w, h = img.size()

                          # 4-3-2


# Show a three color poster
img.show()
```

4-3-1. (2 points) Complete the `luminance()` function.

Note:
input: a tuple with three elements that specify the intensity of red, green, and blue light
output: average luminance value, e.g, input: (60, 80, 160), output: 100

(4-3-1)	

4-3-2. (4 points) Complete the main function to convert an original image to three color poster.

(4-3-2)	

5. (20 points) A permutation of a list is a list that reordered its elements. In this problem, you should implement the function `is_permutation` which takes two lists (`list1` & `list2`) and returns True iif its arguments are permutations of each other or False otherwise.

5-1. (7 points) In the table below, we make test cases for the `is_permutation` function. Fill in the blanks of the table with the return value of the function for each case.

Index	list1	list2	Return Value
1	[]	[]	True
2	[1]	[1]	
3	[1]	[2]	False
4	[1, 2, 3]	[3, 2, 1]	
5	[1, 2, 3]	[4, 5, 6]	
6	[1, 2, 2]	[2, 1, 1]	
7	[1, 2, 3]	[2, 1, 4]	
8	[1, 2, 3]	[3, 2, 1, 2]	
9	[1, 2, 3]	[2, 3, 1, 4]	

5-2. (6 points) One student writes the first version of the function named `is_permuation1`.

```
def is_permutation1(list1, list2):
    for item1 in list1:
        if item1 not in list2:
            return False
    return True
```

However, this function produces wrong return value in some test cases. Explain the reasons with the cases in the 5-1 table.

5-2	

5-3. (7 points) Other pair-programming partner revises the function named `is_permuation2`.

```
def is_permutation2(list1, list2):
    if len(list1) != len(list2):
        return False
    for item1 in list1:
        if item1 in list2:
            list2.remove(item1)
        else:
            return False
    return True
```

This function passes all the test cases in the 5-1 table. However, it has a bad side effect. When we call the `is_permutation2` function, the elements of the second argument list are gone. You can see the example of the code and result in the next table.

Program code	Program result

`li1 = [1, 2, 3]`
`li2 = [3, 2, 1]`
`print(li1)`
`print(li2)`
`print(is_permutation2(li1, li2))`
`print(li1)`
`print(li2)`
	
	`[1, 2, 3]`
	`[3, 2, 1]`
	`True`
	`[1, 2, 3]`
	`[]`

Now, you should implement the perfect/complete version of the `is_permutation` function. It should pass all test cases not only in the 5-1 table but also other kinds of the cases, and has no side effect.

5-3	`def is_permutation(list1, list2):`