Question:	1	2	3	4	5	Total
Points:	20	20	20	20	20	100
Score:						

KAIST CS101 Introduction to Programming
2019 Spring Midterm Exam

Section: ______________________________

Student ID: ______________________________

Name: ______________________________

- Please check if you received all 25 pages of the test material.

  시작하기 전에 반드시 페이지가 총 25쪽인지 확인 하십시오.

- Fill in your student identification number and name. Otherwise, you will lose 1 point for each missing piece of information.

  학번과 이름을 정확히 기입하지 않을 경우, 각 실수 당 1점이 감점됩니다.

- **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 이상이 있을 경우, 왜 문제가 이상이 있다고 생각하는지에 대해 기술하시면 됩니다. 문제가 애매하다고 생각되는 경우 문제를 풀 때 본인이 생각하는 가정을 함께 작성하면 됩니다. 채점 시 가정 및 설명을 고려하겠습니다.

- **Write your answer in Python 3.** We will grade your answers based only on Python 3.

  Python 3를 사용해 문제를 해결하세요. 채점은 Python 3 기준으로만 진행됩니다.

- For multiple choice questions, **check**(✓) in the circle for the correct answer(s).

  객관식 문제의 경우, 올바른 답에 해당하는 동그라미 안에 **체크**(✓)해주세요.

  [Example]

  ✓ Check the correct answer(s).

  ○ Leave other answer(s).

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

1-1. (10 points) Write down the output of the given Python code. If the Python code causes an error, you can simply write down "Error."

1-1-1. `print(3 + 3 ** 2 // 2 % 3)` **4**

1-1-2. `print(2 * "*" * 2)` ****

1-1-3. `print(False or True == False)` **False**

1-1-4. `print([1, 2] + [3, 4])` [1, 2, 3, 4]

1-1-5. `print((1, 2) + 3)` **Error**

1-1-6. `a = [10, 20, 30]`
`print(a[1] - a[-1])` **-10**

1-1-7. `b= [10, 20, 30, 40, 50]`
`print(max(b[1:-2]))` **30**

1-1-8. `l1 = [1, 2]`
`l2 = [1, 2]`
`print(l1 is l2)` **False**

1-1-9. `l3 = (1, 2)`
`l4 = (1, 2)`
`print(l3 == l4)` **True**

1-1-10. `l5 = [1, 2]`
`l6 = [1, 2, 3]`
`print(l5.append(3) == l6)` **False**

1-2. (2 points) For better modularization of a large program, it is recommended to avoid using **global** variables.

○ local
○ tuple
√ **global**
○ list

1-3. (3 points) Please write the result of running the following program. **4**

[Program]

```
sum = 0
for i in range(4):
  for j in range(i):
    sum = sum + j
print(sum)
```

1-4. (5 points) Please write the result of running the following program. [3, 7]

[Program]

```
def sieve(n):
  candidates = list(range(2, n))
  i = 0
  # i: index to a prime number
  while i < len(candidates):
    prime = candidates[i]

    # j: index to a candidate
    #    prime number
    j = i
    while j < len(candidates):
      if candidates[j] % prime == 0:
        candidates.pop(j)
      else:
        j = j + 1
    i = i + 1
  return candidates
print(sieve(10))
```

2. (20 points) Read the following instruction for a programming exercise. Answer the following questions to complete your program code.

   Assume that *cs1robots* library used here is a same version to the library that is provided in the lab session materials on *kaist.elice.io*. You can use all functions of *cs1robots.Robot* like *move()*, *turn_left()*,*front_is_clear()*, *left_is_clear()*, *right_is_clear()*,*facing_north()*, *on_beeper()*, *pick_beeper()*, *drop_beeper()*, or *carries_beepers()*.

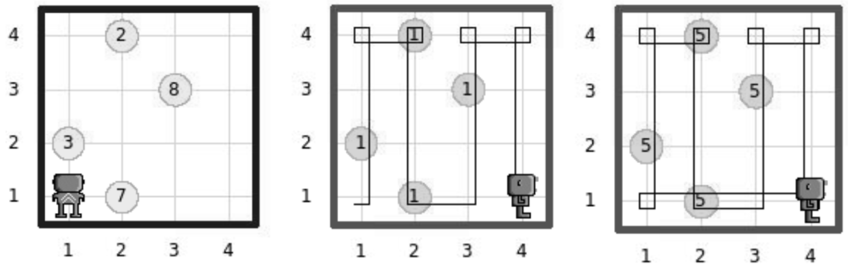
   Each of the six sub-questions (2-1 to 2-6) will be graded independent from each other. The grader will grade a sub-question under the assumption that all other sub-questions are correctly answered.

   [Programming Exercise]
   Given a world that has 4 stacks of beepers, write a code that makes a robot rearrange the beepers in the world so that (1) the 4 stacks of coins remain in the same positions in the world, (2) but the number of coins in each stack are same in the 4 stacks.

   For example, a world that looks like Figure 1-(a), should look like Figure 1-(c) at the end of the program.

   Assume that the world given to you satisfies the following conditions:

   - The world size is n x m, where both n and m are integers larger than or equal to 2.
   - The robot *hubo* is created at the bottom left corner, facing east side.
   - The world has 4 stacks of beepers at random positions.
   - Total number of beepers in the world is divisible to 4.



(a) (b) (c)

Figure 1: Example of a world and stacks of beepers: (a) State of the world after running the line 78, (b) State of the world after running the line 80 and (c) State of the world after running the line 83

[Program]

```
from cs1robots import *

load_world('worlds/mystery_world.wld')

def turn_right():
    for i in range(3):
        hubo.turn_left()

def count_beepers():
    (A)

def move_and_count_beepers():
    hubo.move()
    return count_beepers()

def move_to_wall_and_count_beepers():
    count = 0
    while hubo.front_is_clear():
        count = count + move_and_count_beepers()
    return count

def zigzag_and_count_beepers():
    count = (B)
    while hubo.front_is_clear():
        count = count + move_to_wall_and_count_beepers()
        turn_right()
        if hubo.front_is_clear():
            count = count + (C)
            turn_right()
            count = count + (D)
            hubo.turn_left()
            if hubo.front_is_clear():
                print('checkpoint') # PRINT A MESSAGE
                count = count + (E)
                hubo.turn_left()
    return count

def drop_beepers(count):
    (F)

def move_and_drop_beepers(count):
    hubo.move()
    drop_beepers(count)

def move_to_wall_and_drop_beepers(count):
    while hubo.front_is_clear():
        move_and_drop_beepers(count)

def zigzag_and_drop_beepers(number):
    (G)
    while hubo.front_is_clear():
        move_to_wall_and_drop_beepers(number)
        turn_right()
        if hubo.front_is_clear():
            (H)
            turn_right()
              (I)
            hubo.turn_left()
            if hubo.front_is_clear():
                  (J)
                hubo.turn_left()
    return count

def move_to_origin():
    while not hubo.facing_north():
        hubo.turn_left()
    hubo.turn_left()
    while hubo.front_is_clear():
        hubo.move()
    hubo.turn_left()
    while hubo.front_is_clear():
        hubo.move()
    hubo.turn_left()
    hubo.turn_left()

hubo = Robot()
hubo.set_trace('black')
hubo.turn_left() # Figure-(a)

count = zigzag_and_count_beepers() # Figure-(b)

move_to_origin()
zigzag_and_drop_beepers(   (K)   ) # Figure-(c)
```

2-1. (3 points) Fill in the blank **(A)** to implement a function *count_beepers* that takes no parameters and returns an integer value.

When this function is called, *hubo* should check if it is on a beeper. If *hubo* is not on any beepers, it should do nothing and return 0. If *hubo* is on one or more beepers, *hubo* should pick up all beepers at the position and drop one beeper back to indicate that there was a stack of beepers there, and return the number of beepers that used to be at that position.

For example, if *hubo* is on 5 beepers when the function is called, *hubo* should pick 5 beepers up, drop one beeper, and the function should return 5.

Be aware that calling this function should not move or rotate *hubo*.

[Answer box]

Solution:

```
count = 0
while hubo.on_beeper():
    hubo.pick_beeper()
    count = count + 1
if count > 0:
    hubo.drop_beeper()
return count
```

2-2. (4 + 1 points) Choose the correct answer to fill in the blanks so that the function*zigzag_and_count_bee* makes *hubo* visit the entire world in a zigzag fashion and pick up all but one beepers from every stack of beepers. You can assume that *hubo* is at the bottom left corner of the world facing north when this function is called. You get 1 extra point if you choose the right answers for all questions.

2-2-1. (1 point) Which one should fill the blank **(B)** in?

○ `0`
√ **`count_beepers()`**
○ `move_and_count_beepers()`
○ `move_to_wall_and_count_beepers()`

2-2-2. (1 point) Which one should fill the blank **(C)** in?

○ `1`
○ `count_beepers()`
√ **`move_and_count_beepers()`**
○ `move_to_wall_and_count_beepers()`

2-2-3. (1 point) Which one should fill the blank **(D)** in?

○ `1`
○ `count_beepers()`
○ `move_and_count_beepers()`
√ **`move_to_wall_and_count_beepers()`**

2-2-4. (1 point) Which one should fill the blank **(E)** in?

○ `1`
○ `count_beepers()`
√ **`move_and_count_beepers()`**
○ `move_to_wall_and_count_beepers()`

2-3. (3 points) How many times does `print('checkpoint')` in the function *zigzag_and_count_beepers* prints out the message *checkpoint* if the world shown in Figure 1-(a) is loaded?

[Answer box]

Solution: 1

2-4. (2 points) Fill in the blank **(F)** to implement a function *drop_beepers* that takes one parameter *count* and returns no value.

When this function is called, *hubo* should check if it is on a beeper. If *hubo* is not on any beepers, it should do nothing. If *hubo* is on one or more beepers, *hubo* should drop a beeper for *count* times at the position.

Be aware that calling this function should not move or rotate *hubo*.

[Answer box]

Solution:

```
if hubo.on_beeper():
    for i in range(count):
        hubo.drop_beeper()
```

2-5. (4 + 1 points) Choose the correct answer to fill in the blanks so that the function *zigzag_and_drop_beepers* makes *hubo* visit the entire world in a zigzag fashion and drop *number* (parameter) beepers at the position where there is already one or more beepers. You can assume that *hubo* is at the bottom left corner of the world facing north when this function is called. You get 1 extra point if you choose the right answers for all questions.

2-5-1. (1 point) Which one should fill the blank **(G)** in?

○ `hubo.move()`
√ **`drop_beepers(number)`**
○ `move_and_drop_beepers(number)`
○ `move_to_wall_and_drop_beepers(number)`

2-5-2. (1 point) Which one should fill the blank **(H)** in?

○ `hubo.move()`
○ `drop_beepers(number)`
√ **`move_and_drop_beepers(number)`**
○ `move_to_wall_and_drop_beepers(number)`

2-5-3. (1 point) Which one should fill the blank **(I)** in?

○ `hubo.move()`
○ `drop_beepers(number)`
○ `move_and_drop_beepers(number)`
√ **`move_to_wall_and_drop_beepers(number)`**

2-5-4. (1 point) Which one should fill the blank **(J)** in?

○ `hubo.move()`
○ `drop_beepers(number)`
√ **`move_and_drop_beepers(number)`**
○ `move_to_wall_and_drop_beepers(number)`

2-6. (2 points) Fill in the blank **(K)** to make *hubo* visit the entire world in a zigzag fashion and drop the right number of beepers for each beeper stack.

[Answer box]

Solution: `count // 4 - 1`

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

3-1. (5 points) Write both the *value* and the *type* of the variable `result` after each program in the answer box. If an error occurs in the program, write simply "Error" in the answer box instead.

3-1-1. (1 point) The keyword `break` terminates a loop in which `break` is placed. You get 1 point for answering both the value and the type correctly.

[Program]

```
a = 0
b = True
while True:
    if b:
        a = a + 1
    for i in range(a):
        b = not b
    if a >= 5:
        result = b
        break
```

[Answer box for 3-1-1]

Solution: False, bool

3-1-2. (2 points) You get 1 point for the value and 1 point for the type.

[Program]

```
result = 5
a, b = 3, 4
if a**2 > b*2 and 7 // 2 == 1:
    result = '2'
elif a == b or 1 == float('1.0'):
    result = result - a
else:
    result = b / 2
```

[Answer box for 3-1-2]

Solution: 2, int

3-1-3. (2 points) You get 1 point for the value and 1 point for the type.

[Program]

```
result = '1'
for i in range(3):
    if i % 3 == 1:
        result = result * 2
    else:
        result = result + str(i)
result = int(result) / 2
```

[Answer box for 3-1-3]

Solution: 5051.0, float

3-2. (5 points) Write the ouput from each program in the answer box. If an error occurs in the program, write "Error" in the answer box instead.

3-2-1. (1 point) The input is given as 3. You can write "/" instead of changing line.

[Program]

```
num = input('How_many_apples_do_you_want?_')
for i in range(num):
    print('Peeling_an_apple.')
print('Total_number_of_peeled_apples_is_'+num)
```

[Answer box for 3-2-1]

Solution: ERROR

3-2-2. (2 points) You can write "/" instead of changing line.

[Program]

```
b, c = 4, 5
while b < 20:
    a, b, c = b, c, b+c
    print(c)
```

[Answer box for 3-2-2]

Solution:
9
14
23
37

3-2-3. (2 points) Assume that cs1media library is same as the one you've experienced in the lab session. You can write "/" instead of changing line.

[Program]

```
from cs1media import *


img = create_picture(5, 5)
width, height = img.size()
for y in range(height):
    for x in range(width):
        img.set(x, y, (x+1, 5-x, x**2))
for i in range(3):
    print(img.get(i, i)[i])
```

[Answer box for 3-2-3]

Solution:
1
4
4

3-3. (10 points) Complete the program that produces the desired outcome by filling in one or more blanks in the following programs. You must write only one line of code for each blank.

3-3-1. (4 points) Given program prints reversed digits of `num`. `(I)` can be any integer bigger than zero. Fill in the code for `(A)`.

[Program]

```
num = (I)

result = ''

temp = num
while temp != 0:
    result = (A)
    temp = temp // 10

print(result)
```

[Answer box for 3-3-2 `(A)`**]**

Solution: `result + str(temp % 10)`
or `result + str(temp - (temp // 10) * 10)`

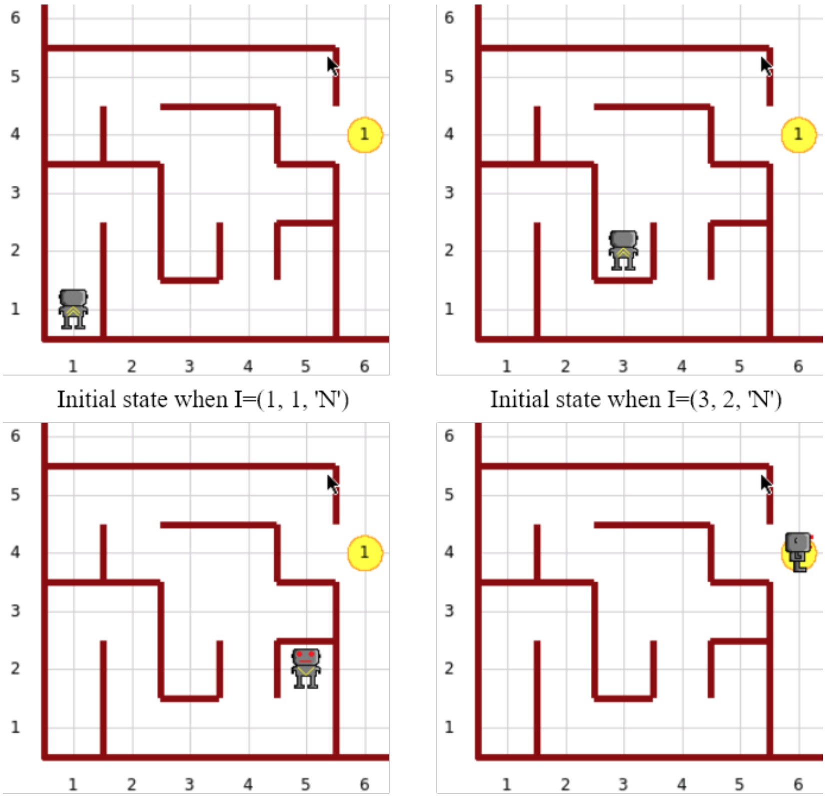
3-3-2. (6 points) The following program makes hubo escape the maze via *left-hand rule*, which follows the wall on the left side. `(I)` is one of tuples: `(1, 1, ‘N’)`, `(5, 2, ‘S’)`, and `(3, 2, ‘N’)`. Fill in `(B)` and `(C)` properly so that the program can produce the target state for any `(I)`. You get 2 points for `(B)` and 4 points for `(C)`.

Assume that `cs1robots` library is same as the one you've experienced in the lab session. The string `‘worlds/maze1.wld’` is a name of the legitimate robot world file that contains the map shown in Figure 2.

[Program]

```
from cs1robots import *


def turn_right():
    for i in range(3):
        hubo.turn_left()

def wall_at_left():
    while not hubo.front_is_clear():
        turn_right()

def escape():
    while hubo.left_is_clear() and hubo.front_is_clear():
        hubo.move()
        if hubo.on_beeper():
                 (B)
    wall_at_left()
    while not hubo.left_is_clear():
        hubo.move()
        while       (C)        :
            hubo.turn_left()
            hubo.move()
            if hubo.on_beeper():
                  (B)
        wall_at_left()

load_world('worlds/maze1.wld')

x, y, ori =   (I)
hubo = Robot(avenue=x, street=y, orientation=ori)

escape()
```

[Initial states and target state]



Initial state when I=(1, 1, 'N')

Initial state when I=(3, 2, 'N')

Initial state when I=(5, 2, 'S')

Target state for any case

Figure 2: Initial states when `(I)` is `(1, 1, 'N')`, `(5, 2, 'S')`, or `(3, 2, 'N')` and the target state of the program

[Answer box for 3-3-2 `(B)`**]**

Solution: `return`

[Answer box for 3-3-2 `(C)`**]**

Solution: `hubo.left_is_clear()`

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

4-1. (5 points) Write the result of the following program. (The result is 4 lines)

[Program]

```
x = 3
y = 5

def f(x, y):
    x = x + y
    if x > 7:
        return x
        x = x - 2
    else:
        x = x * 2
    return x

def g(x):
    y = x + 5
    return x, y

print(f(0, 2))
print(f(x, y))
x, y = g(y)
print(x)
print(y)
```

[Answer box for 4-1]

Solution:
4

Solution:
8

Solution:
5

Solution:
10

4-2. (7 points) In **Homework 1-1**, Christmas Present for Hubo, there were three assumptions as follows:
Assumption 1. The world size is *n* × *m*, where both *n* and *m* are integers from 1 to 10.
Assumption 2. No two stacks will have the same number of beepers.
Assumption 3. There will be at least one stack in each world.

4-2-1. (5 points) The following function, `assumption1()` is implemented to check whether a given world satisfies the Assumption 1 or not. The function returns `True` if the Assumption1 is satisfied. Otherwise it returns `False`. It is assumed that *n* and *m* are integers equal to or greater than 1, and that Hubo is already created as `hubo` by `hubo = Robot(orientation=‘E’, avenue = 1, street = 1).` The function first stores the start position of the Hubo as `x = 1` and `y = 1`. The function then makes the Hubo move to east until it hits the wall, and adds 1 to the `x` each time Hubo steps to east. While Hubo is moving, the function immediately returns `False` when `x` exceeds 10. Likewise, after Hubo meets the wall, the function makes Hubo move to north and adds 1 to the `y`, and immediately returns `Flase` when `y` exceeds 10. At the end of the function, it returns `True`.

[Program]

```
def assumption1():
    x = 1
    y = 1

    while hubo.front_is_clear():
        hubo.move()
        x = x + 1
        if x > 10:
            return [   (A)   ]
    [   (B)   ]
    while hubo.front_is_clear():
        hubo.move()
        y = y + 1
        if y > 10:
            return [   (A)   ]
    return [   (C)   ]
```

Fill in the blanks so that `assumption1()` works properly.

[Answer box for (A)]

Solution:

```
False
```

[Answer box for (B)]

Solution:

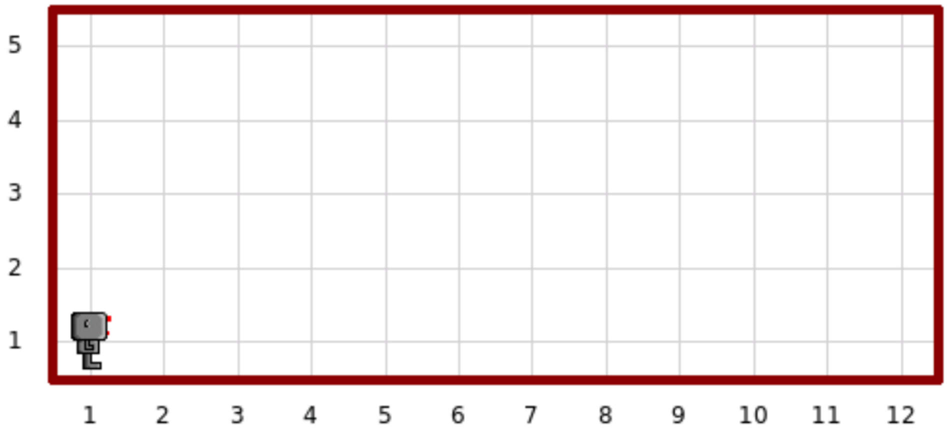
```
hubo.turn_left()
```

[Answer box for (C)]

Solution:

```
True
```

4-2-2. (2 points) When `assumption1()` of the Problem **4-2-1** is executed in the following world, either mark the last position of the Hubo or wirte the coordinate of the position in the form (*x*, *y*).



4-3. (8 points) A **quadratic equation** is any equation having the form

$$ax^2 + bx + c = 0$$

where x represents unknown variable, and a, b, and c are known coefficients of the equation. The number of real roots of the equation can be obtained by the following formula:

$$b^2 - 4ac$$

A quadratic equation can have up to two real roots. If $b^2 - 4ac < 0$, the equation has no real roots. If $b^2 - 4ac = 0$, the equation has only one real root. As a special case, the equation has infinite number of roots when $a = b = c = 0$, and has no root at all when $a = b = 0$ and $c \neq 0$.

The following function, `solve_quad(a, b, c)`, is implemented to find the number of real roots of the quadratic equation $ax^2 + bx + c = 0$. The parameters `a`, `b`, and `c` mean the coefficients of the equation, a, b, and c respectively. The function returns the number of real roots, 0, 1, or 2, of the quadratic equation with given a, b, and c. If the equation has infinite number of roots, the function returns -1.

4-3-1. (5 points) Fill in the blanks so that `solve_quad(a, b, c)` works properly.

[Program]

```
def solve_quad(a, b, c):
    det = b*b -4*a*c
    if det > 0:
        return    (A)
    elif det < 0:
        return    (B)
    else:
        if a == 0 and b == 0:
            if    (C)    :
                return 0
            else:
                return    (D)
        else:
            return    (E)
```

[Answer box for (A)]

Solution:

```
2
```

[Answer box for (B)]

Solution:

```
0
```

[Answer box for (C)]

Solution:

```
c != 0
```

[Answer box for (D)]

Solution:

```
-1
```

[Answer box for (E)]

Solution:

```
1
```

4-3-2. (3 points) Write the result of the following program.
(Note that `solve_quad(a, b, c)` is the same function as in Problem **4-3-1**.)
`[Program]`

```
print(solve_quad(1, 2, 1))
print(solve_quad(0, 0, solve_quad(0, 0, 0)))
print(solve_quad(-1, 2, solve_quad(0, 0, solve_quad(1, 2, 1))))
```

[Answer box for 4-3-2 line 1]

Solution:

```
1
```

[Answer box for 4-3-2 line 2]

Solution:

```

0
```

[Answer box for 4-3-2 line 3]

Solution:

```

2
```

5. (20 points) Answer each question according to the instruction.

5-1. (10 points) Write the result of following codes. If the codes produce an error, explain the reason of the error briefly and state where the error occurred.

5-1-1. (3 points) **[Program]**

```
x = True

def print_x_1():
    print(x)

def print_x_2(x):
    print(x)

def print_x_3():
    x = False
    print(x)

print_x_1()
print_x_2(x)
print_x_3()
```

[Answer box for 5-1-1]

```
Solution: True
True
False
```

5-1-2. (3 points) **[Program]**

```
x = True

def print_x():
    print(x)
    x = False
    print(x)

print_x()
```

[Answer box for 5-1-2]

```
Solution: "UnboundLocalErrorError"
reason: x is a local variable in the function,
because of the assignment,
but has no value inside the first print statement.
```

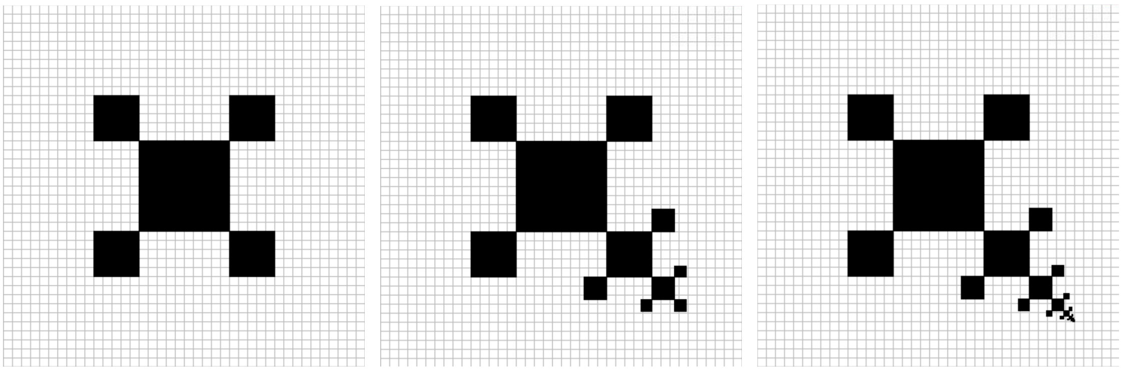
5-1-3. (4 points) **[Program]**

```
def swap(x, y):
    x, y = y, x

def function_1(x, y):
    x + y

def function_2(x, y):
    return x + y

def function_3():
    global x, y
    x += y
    y += x
    return y

def function_4(x, y=5):
    return x + y

x, y = 1, 2
swap(x, y)
function_1(x, y)
x = function_2(x, y)
x = function_3()
x = function_4(x)
print(x, y)
```

[Answer box for 5-1-3]

Solution: `12 7` (*NoComma*)

5-2. (10 points) Complete the following program that draws squares like the following figure by filling in the empty blanks of functions. (Hint: small squares may overlap on large squares)



(a) Depth 1 (b) Depth 3 (c) Depth 10

[Program]

```
from cs1graphics import *

canvas = Canvas(400, 400)

for i in range(40):
    # Draw dashed lines to indicate x, y coordinates
    xline = Path(Point(i*10, 0), Point(i*10, 400))
    xline.setBorderColor('gray')
    canvas.add(xline)
    yline = Path(Point(0, i*10), Point(400, i*10))
    yline.setBorderColor('gray')
    canvas.add(yline)

def draw_black_square(x, y, size):
    # Draw black square using Square class
    square = Square(size, Point(x, y))
    square.setFillColor('black')
    canvas.add(square)

def draw_square_four_ears(x, y, s):
    # Draw 4 squares which sizes are s/2,
    # for each ear of the square at (x, y) and size s
    [                 (5-2-1)                 ]

# Draw squares within given depth
x, y, size, depth = 200, 200, 100, 10

# Draw the largest square as base
draw_black_square(x, y, size)

for d in range(depth):
    [                 (5-2-2)                 ]

```

5-2-1. (5 points) Complete the **draw_square_four_ears** function by filling in the box.
[Answer box for 5-2-1]

Solution:

```
        draw_black_square(x+s*3/4, y+s*3/4, s/2)
        draw_black_square(x-s*3/4, y+s*3/4, s/2)
        draw_black_square(x+s*3/4, y-s*3/4, s/2)
        draw_black_square(x-s*3/4, y-s*3/4, s/2)

```

5-2-2. (5 points) Complete the for loop that draws squares by filling in the box.
[Answer box for 5-2-2]

Solution:

```
        draw_square_four_ears(x, y, size)
        x, y = x+size*3/4, y+size*3/4
        size = size * 1/2

```