| 1<br>( 20 ) | 2<br>( 20 ) | 3<br>( 20 ) | 4<br>( 20 ) | 5<br>( 20 ) | TOTAL<br>(100) |
|---|---|---|---|---|---|
| | | | | | |

# [ CS101 ] Introduction to Programming
# 2014 Spring - Midterm Examination

| SECTION | STUDENT ID | NAME |
|---|---|---|
| | | |

※ Please check to see if you have all **17 pages** in your test material.
※ 시작하기 전에 반드시 페이지의 수를 확인 하십시오. ( 전체 : **17 쪽** )

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

※ **We will not answer questions about the problems during this exam**. If you think there is anything ambiguous, unclear or wrong about a problem, please write down your reasons and make necessary assumptions to proceed with the problem. We will take your explanation into consideration when 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 following instruction.

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

```
aseq = ["CS101", "A+", 13]
aseq.reverse()
print aseq[-3:-1]
```

_____

**1-2. (4 points)** Compare two operators '=' and '=='. Describe how they are different.

'=' is used for _____,

but '==' is used for _____.

**1-3. (2 points)** *'raw_input'* function prints out a message which is given as a parameter (typed in 'string') and waits for a user to enter a sequence of letters (or characters) using a keyboard. When the user presses the 'Enter' key, the sequence of letters is returned as a message (typed in 'string').

Following the description of '*raw_input*' function given above, choose a number which shows the correct result of the following program.

( ※ The user's input is denoted as **bold type** in the results. )

**[ Program ]**

```
print raw_input(raw_input("Ultra!"))
```

**[ Result ]**

| ① | ② | ③ | ④ |
|---|---|---|---|
| Ultra!**Rapid!**<br>Ultra!Rapid!**Fire!**<br>Fire! | **Rapid!**<br>**Fire!**<br>Fire! | Ultra!**Rapid!**<br>Ultra!**Fire!** | Ultra!**Rapid!**<br>Rapid!**Fire!**<br>Fire! |

**[ Correct result ]**

_____

**1-4. (2 points)** The following two programs are slightly different as denoted in **bold type**.

**[ Program 1 ]**

```
def divisor(a):
    if a%3 == 0:
        return "3"
    elif a%2 == 0:
        return "2"
print divisor(6)
```

**[ Program 2 ]**

```
def divisor(a):
    if a%3 == 0:
        return "3"
    if a%2 == 0:
        return "2"
print divisor(6)
```

Do those two programs return the same results? ( Answer with a 'Yes' or 'No'. ) _____

**1-5. (10 points)** For each statement, please answer with "T" if the answer is true or answer with "F" for false. ( If you don't know the answer for a statement, you can also leave it blank. )

If your answer is correct, then you will get 1 point. Otherwise, you will lose 1 point.
( ※ A blank will be counted as **no point**. )

| Statement | Answer |
|---|---|
| (Example) Variables defined outside of a function are called ***global variables***. | T |
| **Modularization** means that software consists of parts that are developed and tested separately. | |
| A **function** cannot be an argument of another function. | |
| In a function, **parameters** can be used as **global variables**. | |
| Lists and tuples are very similar, but lists are **mutable**, while tuples are **immutable**. | |
| **Semantic error** is a kind of error that results in no error message but the program does not do what it is supposed to do. | |
| A **while-loop** repeats instructions a fixed number of times. | |
| A condition is something that is either **True** or **False**. | |
| A function that does not execute **return** automatically returns **None**. | |
| When a function is called, the arguments of the function are assigned to the parameters. And a function can only return one value. So, a **return** statement such as "**return name, sum_of_scores**" causes an error because the **return** statement tries to return two values. | |
| The function **range()** returns a list of integers. It means **range(4)** returns a list [0, 1, 2, 3]. So **tuple(range(4).reverse())** gives you **Error**. | |

**2. (20 points) Hubo wants Ami to follow the path of his movement. Hubo tells Ami in which direction he moved by dropping beepers, so that Ami can find out the direction by counting the beepers and follow his trace. Hubo uses "the number of beepers" as a sign to tell Ami to which direction he moved.**

**Hubo's signs are as follows: one beeper means to take a left turn and move one step forward, two beepers mean to take a move forward, and three beepers mean to take a right turn and move one step forward.**

**Answer each question according to the following instruction.**

**[ Program ]**

```
from cs1robots import *

create_world() # create_world(4,4) in 2-3-1, create_world(3,3) in 2-3-2

hubo = Robot(beepers=400)
ami = Robot("light_blue")

def hubo_turn(a):
    for i in range(a):
        hubo.turn_left()

def stepback_and_restore():
    #2-1 ( Complete stepback_and_restore function
           that will be used inside move_one_step_or_stay function. )

def follow_trace():
    #2-2 ( Complete follow_trace function
           that will be used for Ami to follow Hobo's trace. )

while can_move():
    move_one_step_or_stay()

follow_trace()
```

```
def can_move():
    retVal = False

    if hubo.front_is_clear():
        print "A",
        hubo.move()
        if not hubo.on_beeper():
            retVal = True
        hubo_turn(2)
        hubo.move()
        hubo_turn(2)
        if retVal:
            print ""
            return retVal
    if hubo.left_is_clear():
        print "B",
        hubo_turn(1)
        hubo.move()
        if not hubo.on_beeper():
            retVal = True
        hubo_turn(2)
        hubo.move()
        hubo_turn(1)
        if retVal:
            print ""
            return retVal
    if hubo.right_is_clear():
        print "C",
        hubo_turn(3)
        hubo.move()
        if not hubo.on_beeper():
            retVal = True
        hubo_turn(2)
        hubo.move()
        hubo_turn(3)
        if retVal:
            print ""
            return retVal

    return retVal
```

```
def move_one_step_or_stay():
    if hubo.front_is_clear():
        a = 2
        hubo.move()
        if hubo.on_beeper():
            a = 1
        hubo_turn(2)
        hubo.move()
        hubo_turn(2)
    else:
        a = 1

    if a == 1:
        if hubo.left_is_clear():
            for i in range(a):
                hubo.drop_beeper()
            hubo_turn(a)
            hubo.move()
            if hubo.on_beeper():
                stepback_and_restore()
    elif a == 2:
        if hubo.front_is_clear():
            for i in range(a):
                hubo.drop_beeper()
            hubo.move()
            if hubo.on_beeper():
                stepback_and_restore()
    elif a == 3:
        if hubo.right_is_clear():
            for i in range(a):
                hubo.drop_beeper()
            hubo_turn(a)
            hubo.move()
            if hubo.on_beeper():
                stepback_and_restore()
```

**2-1. (5 points)** Hubo moves to a new position using ***move_one_step_or_stay*** function. However, if there are beepers at the new position, you have to make the robot come back to the previous position; that is, at the same position and with the same orientation. This role is handled by the function ***stepback_and_restore***. Implement ***stepback_and_restore*** method.
( ※ This corresponds to **#2-1** blank box. )

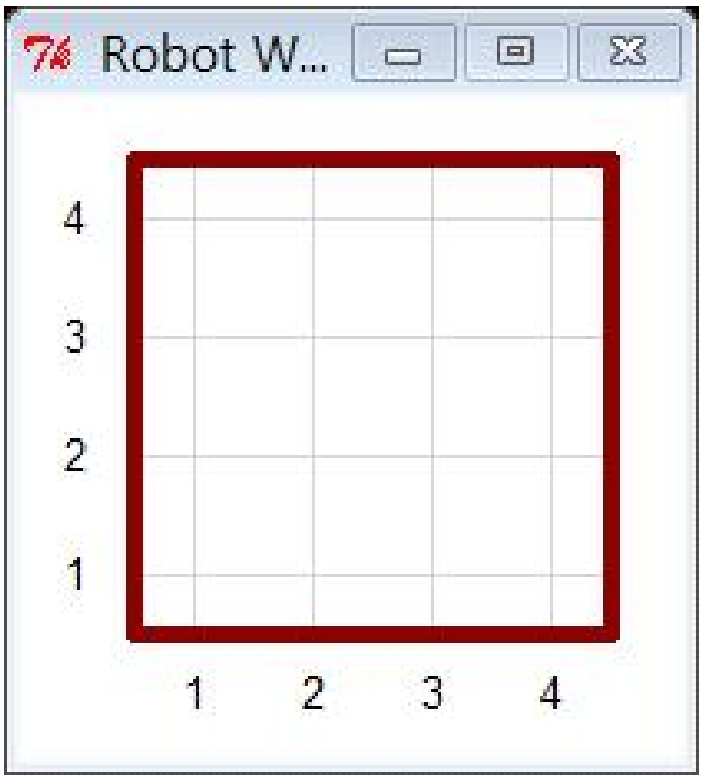
```
def stepback_and_restore():
```

**2-2. (5 points)** Write the function ***follow_trace*** for Ami. A robot picks and counts the beepers, and moves to the proper direction while a robot is on beepers.
( ※ This corresponds to **#2-2** blank box. )

```
def follow_trace():
```

**2-3. (10 points)** In current ***move_one_step_or_stay*** function, Hubo does not show any random movement like HW1, and there are several ***print*** statements added in ***can_move*** function. Answer each question below.

**2-3-1. (5 points)** If the world's size is **4x4**, draw the Hubo's movement by writing the number of beepers like ①, ②, or ③. The final position of Hubo can be drawn as ■. Do not consider the direction of Hubo in the final position.




**2-3-2 (5 points)** There are several ***print*** statements added in ***can_move*** function. If the world's size is **3x3**, write the output of the entire program. Be aware the usage of "," in ***print*** statements.

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

**3-1 (5 points)** You are asked to implement ***is_positive*** function which determines a number is positive or not. ***is_positive*** function does not return any value but it prints 'Yes' when parameter x is positive, otherwise it prints 'No'.
( ※ Hint : 0 is not a positive number. )

**[ Program ]**

```
def is_positive(x):
    
```

**is_positive(2.3)**
**is_positive(-3)**
**is_positive(0)**

**[ Result ]**

**Yes**
**No**
**No**

**3-2 (5 points)** You are asked to implement ***power*** function which prints parameter ***x*** to the parameter ***y***. The ***power*** function does not return any value but prints the result of calculation $x^y$. The parameter ***y*** can be positive integer or 0, hence, you must check this condition in the ***power*** function. You cannot use '**' operator in ***power*** function.

**[ Program ]**

```
def power(x,y):
    
power(2,2)
power(2,3)
power(3,2)
power(3,0)
power(3,-2)
```

**[ Result ]**

```
4
8
9
1
Error : parameter y must be positive integer or 0
```

**3-3 (10 points)** You are asked to implement a program which prints all prime numbers less than parameter ***n***. The program also prints the number of prime numbers less than ***n***. You must use ***is_prime*** function in ***print_prime*** function to determine a number is prime or not.

**[ Program ]**

```
def is_prime(x):
    
    
def print_prime(n) :
    
    
print_prime(20)
```

**[ Result ]**

```
2
3
5
7
11
13
17
19
The number of prime numbers less than 20 is 8
```

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

**4-1. (2 points)** What is the result of the following program?

```
hubo = "Robot()"
if type(hubo) != type("True"):
    print "HUBO"
else:
    print "AMI"
```

_____

**4-2. (2 points)** What is the result of the following program?

```
var = 0

def MyFunc1():
    var = 10

def MyFunc2():
    global var
    var = 3.14

MyFunc1()
print var
MyFunc2()
print var
```

_____

_____

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

```
def swap(x, y):
    x = y
    y = x
    return x, y
x = 10
y = 20
x, y = swap(x, y)
print x * y
```

_____

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

```
var = 5

def MyFunc1():
    var = 10

def MyFunc2():
    global var
    var = str(var) * 2
    return var

if type(MyFunc1()) == type(3.14) and type(MyFunc2()) == type("101") :
    print var * 2
else :
    print var
```

**4-5. (4 points)** What is the result of the following program?

```
def MyFunc1(var1, var2) :
    if len(var1) > len(var2) :
        return var1 * len(var2)
    return var2 + var1

def MyFunc2(var1, var2) :
    result = ""
    while var1 < var2 :
        result = var1[0] + result + var2[-1]
        var1 = var1[1:-1]
        var2 = var2[1:-1]
    return result

print MyFunc1("CS", "101") + MyFunc1("101", "CS")
print MyFunc2("HELLO", "WORLD")
```

**4-6. (6 points)** Define two functions ***mean*** and ***variance***.

( ※ You must call the function ***mean*** in the function ***variance***. )

The function ***mean*** takes three float values and returns their arithmetic mean. The arithmetic

mean is the sum of the sampled values divided by the number of items in the sample:

$$\overline{x} = \frac{x_1 + x_2 + ... + x_n}{n}$$

※ IN (arguments) : three float values / OUT (result) : a float value

The function ***variance*** takes three float values and prints out their variance. The variance is the expected value of the squared deviation from the mean:

$$\sigma^2 = \frac{\left(x_1 - \overline{x}\right)^2 + \left(x_2 - \overline{x}\right)^2 + ... + \left(x_n - \overline{x}\right)^2}{n}$$

※ IN (arguments) : three float values / OUT (result) : none

```
def mean(               ):

def variance(               ):
```

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

**The following program is to perform addition on an arbitrary number of values represented by list objects.**

**[ Program ]**

```
def convert_int_to_list(value):
   result = []
      #5-1 (convert integer value 'value' to list 'result')
   return result

def compare_list(list1, list2):
      #5-2 (complete compare_list function
            that will be used inside addition function)

def handle_carry(list_value):
   index = 0
   while index  < len(list_value):
      index = index + 1
      if list_value[-index] >= 10:
         carry = list_value[-index] / 10
         list_value[-index] = list_value[-index] % 10
         if index >= len(list_value):
            list_value.insert(0, carry)
         else:
            list_value[-index-1] = list_value[-index-1] + carry
   return list_value

def addition(operands):
   operands.sort(compare_list)
   result = operands[0]

   for i in range(1, len(result)+1):
      for j in range(1, len(operands)):
            #5-3 (add all the operands)
   result = handle_carry(result)
   return result
```

**Functions used in the above program are defined as follows:**

| Function | Description |
|---|---|
| **convert_int_to_list** | convert decimal input to list representation |
| - parameters | **value**: positive integer (decimal number) |
| - return value | list object representing the input parameter, **value** |
| - example | >>> **convert_int_to_list**(1328)<br>[1, 3, 2, 8] |
| **compare_list** | comparison method to be used in (list) sort method |
| - parameters | **list1**: list object<br>**list2**: list object |
| - return value | -1, 0, or 1 implying which list should be preceeded. comparison should be performed based on the length of the lists |
| - example | >>> a = [[3], [1, 8]]<br>>>> a.sort(**compare_list**)<br>>>> a<br>[[1, 8], [3]]<br>>>> b = [[3, 1], [1, 8]]<br>>>> b.sort(**compare_list**)<br>>>> b<br>[[3, 1], [1, 8]] |
| **handle_carry** | handle carry for each element of the list |
| - parameters | **list_value**: list object whose elements are integers |
| - return value | list object representing after handling carries such that for any element $\alpha$, $0 \leq \alpha < 10$ |
| - example | >>> **handle_carry**([1, 2, 3])<br>[1, 2, 3]<br>>>> **handle_carry**([12, 34, 56, 7])<br>[1, 5, 9, 6, 7] |
| **addition** | perform addition on arbitrary number of operands |
| - parameters | **operands**: list of lists to be added |
| - return value | list object representing summation of **operands** |
| - example | >>> **radix_addition**([[1], [1, 2], [4, 5]])<br>[5, 8]<br>>>> **radix_addition**([[5, 6], [7, 8]])<br>[1, 3, 4] |

**There are some methods available for a list object *L*.**

| Methods | Description |
|---|---|
| **L**.append(**v**) | add object **v** at the end of **L** |
| **L**.insert(**i**,**v**) | insert element **v** into **L** at position **i** |
| **L**.pop() | remove and return last element of **L** |
| **L**.pop(**i**) | remove and return element of **L** at position **i** |
| **L**.remove(**v**) | remove first element equal to **v** |
| **L**.index(**v**) | return index of first element equal to **v** |
| **L**.count(**v**) | return number of elements equal to **v** |
| **L**.extend(**K**) | append all elements of sequence **K** to **L** |
| **L**.reverse() | reverse the list **L** |
| **L**.sort() | sort the list **L** |

**5-1. (5 points)** Fill in **#5-1** blank box to convert decimal number into list representation.

**5-2. (5 points)** Define ***compare_list*** method such that a list whose elements are list objects is sorted by the length of lists in descending order when 'sort(compare_list)' is called. Notice that your ***compare_list*** method should work on the *length of list objects* as in the above example.

( ※ This corresponds to **#5-2** blank box. )

( ※ Hint : Refer to below ***cmp*** method usage, which is pre-defined method and can be used in sort method (by default) for a list of numbers in ascending order. )

Reference. <Usage of **cmp** method>

```
>>> cmp(1, 3)
-1
>>> sample_list = [1, 5, 3, 7, 9, 0, 3]
>>> sample_list.sort(cmp) # or just sample_list.sort()
>>> sample_list
[0, 1, 3, 3, 5, 7, 9]
```

**5-3. (5 points)** Fill in **#5-3** blank box to add all the elements of the given list.
( **※ Caution: In your answer, you MUST NOT use 'break' or 'continue' !** )

( ※ Hint : Elements of operands can have arbitrary lengths.
Therefore, it is easy to add elements of operands from end to beginning.
i.e., add from least significant digit (LSD) to most significant digit (MSD).
For the case of [1, 2, 3], 1 is MSD and 3 is LSD. )

**5-4. (5 points)** Define a function, named ***restore***, to convert a list representing a decimal number back to the decimal number as follows:

| Function | Description |
|---|---|
| **restore** | convert a value represented as list to decimal number |
| - parameters | **value**: list object representing a decimal number |
| - return value | decimal number corresponding to **value** |
| - example | >>> **restore**([1, 0, 1])<br>101<br>>>> **restore**([5, 6, 7, 8, 9, 0])<br>567890 |

Notice that the function ***restore*** should not be used inside the answers of **#5-1~#5-3**.