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

[CS101] Introduction to Programming
2016 Spring - Final Examination

SECTION	STUDENT ID	NAME

※ Please check if you have all 20 pages of the test material.
※ 시작하기 전에 반드시 페이지의 수를 확인 하십시오.(전체 : 20쪽)

※ 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. (4 points/1 point each) What will the last statement output? Answer "True" or "False".

Statement	Answer

`x = [100, 200, 300]`
`y = x`
`y[1] = 1000`
`x[1] is y[1]`	

`x = (100, 200, 300)`
`y = x`
`x[1] is y[1]`	

`x = ("100", "200", "300")` `len(x) is len(x[2])`	

`x = ("100", "200", "300")`
`y = x`
`x = x + ("400", "500")`
`x[1] is y[1]`	

1-2. (6 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 (6) and errors (a) to (f) are 1:1 mapped.**)

①	②

`x = ("CS101", "A+", "Happy")`
`y[2] = "Happy After Exam"`
	`x = "100"`
	`y = (200, 300)`
	`z = x + y`

③	④
`x = [1,2,3,4,5]` `x[0) = 0`	`x = ("CS101", "A+", "Happy")` `x[3] = "After Exam"`

⑤	⑥

`x = (1,2,3,4,5)`
`x.reverse()`
	`x = [0,1,2,3,4,5,6,7,8,9]`
	`x[1:] = x[:9] + [100]`
	`print( x[9]/x[0])`

(a) SyntaxError **(b)** TypeError **(c)** IndexError
(d) NameError **(e)** AttributeError **(f)** ZeroDivisionError

1-3. (10 points/2 points each) What will the last statement in the following pieces of code output?

Statement	Answer
(1) x = [1,2,3,4] x.append((1,2,3,4)) print(x)	
(2) x = [3,1,4,2] x.sort() print(x)	

(3)
def make_sun(r=10, c="red"):
global sun
if (r == 10):
sun = "yellow"
else
sun = c
 	
make_sun(20, "bloody")
print(sun)	

(4) x = [0,1,2,3] x[1:] = x[:4] + [100] print(x)	

(5)
def Ack(m = 0, n = 0):
if (m == 0):
return n+1
elif (m>0 and n == 0):
return Ack(m-1,1)
else
return Ack(m-1, Ack(m, n-1))
 	
print (Ack(3,3))	

2. (20 points) Answer according to the instruction.

Below you will be given a file named **'problems1.txt'.** It contains pairs of questions and answers, and each pair is separated by a blank line. In the rest of this problem, you will be asked to complete **readQuiz()** and implement **doAnswer().** Lastly, you should complete the piece of code that computes a correct answer ratio based on questions given by **'problems2.txt'.**

['problems1.txt']

```
A ____-loop repeats certain instructions a fixed number of times
for

Object provides ____ to perform certain actions
methods

____ functions convert objects from one type to another type
Type conversion
                                    ⋮
```

[Program Code]

```
import random

qnaList = []

# Read questions and answers from the file.
def readQuiz(filename) :
    f = open(filename, "r")
    lines = f.readlines()

    questions = []
    answers = []
    
    2-1

    return questions, answers

# Memorize function
def doMemorize(questions, answers) :
    for i in range(len(questions)) :
        qnaList.append([questions[i], answers[i]])

# Answer function
def doAnswer(question) :
    
    2-2
```

2-1 (8 points) Complete the function **readQuiz()** above. It reads all pairs of the questions and answers from the file **'problems1.txt'**. This function should return two string lists that contain the question and answer strings, respectively.

※ Note that there is a blank line between each of the pairs in **'problems1.txt'**.

2-2 (6 points) Implement the function **`doAnswer().`** It receives a question string as an input parameter and returns the corresponding answer string based on the function **`doMemorize().`** For example, if the question string is "Object provides ____ to perform certain actions", the function should return the string "methods".

```
def readQuiz(filename):
### Assume it is implemented correctly

def doAnswer(question) :
### Assume it is implemented correclty

# ******************** Test main routine ********************
questions, answers = readQuiz("problems1.txt")
doMemorize(questions, answers)

newQ, newA = readQuiz("problems2.txt")
numQuestion = len(newQ)

score = 0
for i in range(numQuestion) :
```

2-3

```
    else :
        print ("-------------- Your answer is wrong! -------------")
        print ("The questions is ... ")
        print (newQ[i])
        print ("The answer based on 'problems2.txt' is [%s]" % newA[i])
        print ("But the correct answer based on 'problems1.txt' is [%s]" % ans)

ratio = score / float(numQuestion)
print ("score : %d / %d (%d%%)" % (score,  numQuestion, int(ratio * 100)))
```

[Example of Result in one wrong answer case]

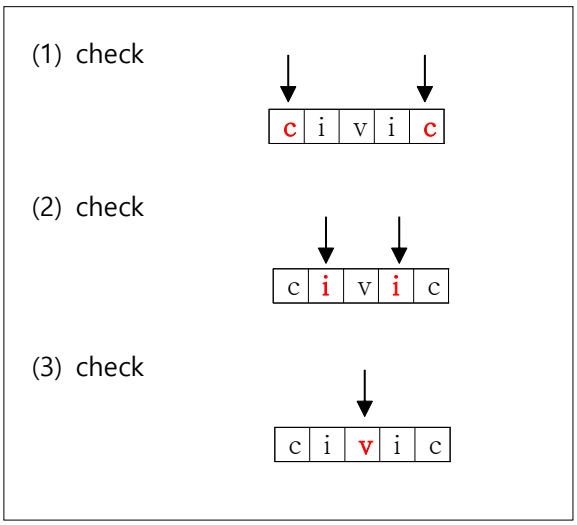
```
  -------------- Your answer is wrong! -------------
  The questions is ...
  A _____-loop repeats instructions as long as some condition is true
  The answer based on 'problems2.txt' is [forever]
  But the correct answer based on 'problems1.txt' is [while]

  score : 24 / 25 (96%)
```

2-3 (6 points) Now assume that you have implemented **`readQuiz()`** and **`doAnswer()`** correctly. You are given another file named **`'problems2.txt'`**. It has the same format as **`'problems1.txt'`**. The questions in **`'problems2.txt'`** is a subset of the questions in **`'problems1.txt'`**. However, for the same question, **`'problems2.txt'`** may have a different answer from **`'problems1.txt'`**. Complete the code so that it will output as in the example, given that **`'problems2.txt'`** has 25 pairs.

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

3-1 (4 points) A palindrome is a word that is spelled same both forward and backward. Some examples of palindromes are "civic", "rotator", "madam" and "kayak". The following function `palindrome()` returns `False` if the word is not a palindrome, and returns `True` if the word is a palindrome.

The strategy is to compare character by character from each end as demonstrated below.



What should be in the blank in lines 3 and 4?

```
1 def palindrome(word):
2     end = len(word)-1
3     for i in          (3-1-1)
4         if            (3-1-2)
5             return False
6     return True
7
```

(3-1-1) _____ (2 points)

(3-1-2) _____ (2 points)

3-2 (6 points) What is the result of the following code? (lines 16 and 17)

```
1  a=12
2  b=34
3
4  def swap(a, b):
5      b, a = a, b
6      return a, b
7
8  w = 56
9  x = 78
10 y = 910
11 z = 1112
12
13 result = swap(swap(w, x), swap(y, z))
14
15 print (type(swap)) #(3-2-1)
16 print (type(result)) #(3-2-2)
17 print (result)      #(3-2-3)
```

(3-2-1)_____ (2 points)

(3-2-2)_____ (2 points)

(3-2-3)_____ (2 points)

3-3 (10 points) Answer the questions.

```
num_list = [4,2,8,6,0]

def swap2(num_list, a, b):
    tmp = num_list[a]
    num_list[a] = num_list[b]
    (3-3-1)

def mystery_function1(num_list):
    for i in range(len(num_list)):
        for k in range(len(num_list)-1, i, -1):
           if (num_list[k] < num_list[k-1]):
        swap2(num_list, k, k-1)
    return num_list

def mystery_function2():
  num_list=[3,9,7,5,1]
  for index in range(1,len(num_list)):
  currentvalue = num_list[index]
  position = index
  while position > 0 and num_list[position-1] > currentvalue:
num_list[position]=num_list[position-1]
          position = position-1
      num_list.pop(position)              #(3-3-2)
      num_list.insert(position, currentvalue) #(3-3-2)
  global num_list
  print(num_list)

print(mystery_function(num_list)) #(3-3-3)
mystery_function2()            #(3-3-4)
```

3-3-1 Function `swap2()` swaps the two elements, a^{th} element and b^{th} element in a list. What should be in the blank in line 6?

(3-3-1)_____(2 points)

3-3-2 The list operators in lines 22 and 23 a bit cumbersome. Replace them with one line of code that will result in the same as those two lines.

(3-3-2)_____(2 points)

3-3-3 What is the result of the code in line 28?

(3-3-3)_____(2 points)

3-3-4 What is the result of the code in line 29?

(3-3-4)_____(2 points)

3-3-5 Explain what `mystery_function1()` does in one or two sentences in either Korean or English.

(3-3-5)_____

_____

(2 points)

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

4-1. (6 points) What is the result of the following program?

<The_Door.txt>

```
We need Hodor.
Hodor.
Hodor.
Warg into Hodor now! Bran, wake up.
We need Hodor.
We need Hodor.
Warg into Hodor now! Now! Listen to your friend, Brandon.
Hodor: Hodor.
Hodor.
Hodor.
Hodor.
Hodor.
Hodor.
```

※ Note: line separator '\n' is not visible in text editor like Notepad.

```
f = open('The_Door.txt')
for l in f:
  if 'now' in l:
      say = l
      break
f.close()
print(say+'We need Hodor.')
```

(4-1)	

4-2. The following text file contains yearly statistics of the density of atmospheric particulate matter. Data from each year is wrapped with a string tag '<row>' at the beginning and '</row>' at the end. The year is wrapped with '<YYYY>' and '</YYYY>', the number of the fine dust warning alerts is wrapped with '<CNT>' and '</CNT>', and the number of days that reached the threshold density of atmospheric particulate matter is '<DAY_CNT>' and '</DAY_CNT>'. Lastly, the maximum density of atmospheric particulate matter is tagged with '<MAXPPM>' and '</MAXPPM>'.

(MAXPPM meaning maximum density of the atmospheric particulate matter - μg/m³/hour)

<YearlyPM10.txt>

```
<row>
<YYYY>2007</YYYY>
<CNT>2</CNT>
<DAY_CNT>4</DAY_CNT>
<MAXPPM>306</MAXPPM>
</row>
<row>
<YYYY>2008</YYYY>
<CNT>2</CNT>
<DAY_CNT>3</DAY_CNT>
<MAXPPM>304</MAXPPM>
</row>

(skip)...

<row>
<YYYY>2015</YYYY>
<CNT>3</CNT>
<DAY_CNT>5</DAY_CNT>
<MAXPPM>245</MAXPPM>
</row>
```

※ Note: line separator '\n' is not visible in text editor like Notepad.

4-2-1. (7 points) Complete the function `getMaxPPM()` such that it outputs the results shown below.

4-2-1

```
def getMaxPPM(filename):
    f = open(filename,'r')
    mylist = []
    for l in f:
        line = l.strip()

        #4-2-1 ( Complete getMaxPPM()
                such that it returns a list of yearly Max PPMs )

    f.close()
    return mylist
max_ppms = getMaxPPM('YearlyPM10.txt')
print(max_ppms)
```

TERMINAL

```
['306', '304', '288', '270', '0', '0', '304', '192', '245']
```

(Above **['288', '270', '0', '0', '304', '192']** *were skipped in the 'YearlyPM10.txt' file)*

(4-2-1)	

4-2-2. (7 points) Complete `createMaxPPM()` such that it creates a file with .csv extension and has lines of the MAX PPM and the year as in the example below.

※ Note: Don't forget to consider line separator '\n' when you write to a file.

4-2-2

```
def createMaxPPM_File(filename):
    ppm_list = getMaxPPM('YearlyPM10.txt')
    year = 2007
    f = open(filename, 'w')
    for ppm in ppm_list:
        #4-2-2 ( Complete createMaxPPM
                such that it creates a file that contains lines
        of the Max PPM and the year. )
    f.close()

createMaxPPM_File('YearlyPM10.csv')
```

YearlyPM10.csv

```
2007,306
2008,304
2009,288
2010,270
2011,0
2012,0
2013,304
2014,192
2015,245
```

(4-2-2)	

5. (20 points) This program abstracts banks and accounts by defining `Bank` and `Account` classes. Answer each question according to the instruction and complete the program.

Program code

```
class Bank:
  def __init__(self, name):
    self.account_list = []
    self.name = name
  def add_account(self, id, bal):
    acc = Account(id, bal, self.name)
    self.account_list.append(acc)
  def get_account(self, accid):
    for account in self.account_list:
      if account.id == accid:
        return account
    return None
```

 # 5-1-1

```
class Account:
  def __init__(self, accid, bal, bna):
    self.balance = bal
    self.id = accid
    self.bankname = bna
  def deposit(self, money):
    # Deposit money
```

 # 5-2

```
  def withdraw(self, money):
    # Withdraw money
```

 # 5-3

```
  def transfer(self, tobank, toaccid, money):
    # Transfer money
```

 # 5-4

 # 5-1-2

```
WR = Bank("WooRi")
NH = Bank("NeoHui")
WR.add_account(1, 100)
NH.add_account(1, 150)
NH.add_account(2, 10000)
woori_1 = WR.get_account(1)
neohui_1 = NH.get_account(1)
neohui_2 = NH.get_account(2)

print("Banks: %s & %s" %(WR,NH))
print(woori_1)
print(neohui_1)

print("\n# Deposit 500 into WooRi account 1")
woori_1.deposit(500)

print("\n# Withdraw 200 from NeoHui account 1")
neohui_1.withdraw(200)
print("# Withdraw 50 from NeoHui account 1")
neohui_1.withdraw(50)

print("\nTransfer 5000 from NeoHui account 2 to WooRi account 1")
neohui_2.transfer(WR, 1, 5000)
print("# Transfer 1000 from NeoHui account 1 to WooRi account 1")
neohui_1.transfer(WR, 1, 1000)
print("# Transfer 10 from NeoHui account 1 to WooRi account 2")
neohui_1.transfer(WR, 2, 10)
```

Column 1 | Column 2

Program result
Banks: WooRi & NeoHui Bank: WooRi, Account ID: 1, Balance: 100 Bank: NeoHui, Account ID: 1, Balance: 150 # Deposit 500 into WooRi account 1 Result - WooRi account 1's balance: 100 -> 600 # Withdraw 200 from NeoHui account 1 You can't withdraw 200 from NeoHui account 1 # Withdraw 50 from NeoHui account 1 Result - NeoHui account 1's balance: 150 -> 100 Transfer 5000 from NeoHui account 2 to WooRi account 1 Result - NeoHui account 2's balance: 10000 -> 5000 Result - WooRi account 1's balance: 600 -> 5600 # Transfer 1000 from NeoHui account 1 to WooRi account 1 You can't withdraw 1000 from NeoHui account 1 So you can't transfer 1000 from the account # Transfer 10 from NeoHui account 1 to WooRi account 2 WooRi bank doesn't have account 2

Note (You will get zero point if you do not follow these rules.)
1. The code in Column 2 of **`Program code`** and the output in **`Program result`** are just for demonstration. Your answer should run any code that meets the requirements.
2. Do **not** use the import statement.

5-1. (4 points) When you use `Bank` or `Account` objects as arguments to `print()`, Python prints a specific string that depends on the object's properties. Fill in the blanks **5-1-1** and **5-1-2** according to the **`Program result`**.
Hint: Define the special method of the class.

(5-1-1)	
(5-1-2)	

5-2. (4 points) Complete the `deposit()` method in the `Account` class. This method should deposit the amount of `money` into the self account object, and print the result according to the **`Program result`**.

Note:

1. The account `balance` must remain greater than or equal to 0, with no maximum limit.
2. The value of the argument `money` is a positive integer.
3. You have to use string formatting operators to print the result.
   Do not concatenate strings.

(5-2)	

5-3. (5 points) Complete the `withdraw()` method in the `Account` class. This method should withdraw the amount of `money` from the self account object, and return the money value. If the account's balance is less than `money`, return `None.` The method should print the result according to the **`Program result`**.

Note:

1. The three conditions in **5-2** apply the same in this problem.

(5-3)	

5-4. (7 points) Complete the `transfer()` method in the `Account` class. This method should transfer the amount of `money` from the self account object to the destination account whose id is `toaccid` in `tobank.` First, check the existence of the destination account, and then withdraw the amount of `money` from the self account object. If there is no problem, deposit the amount of `money` in the destination account. If problems occur, you should print the message according to the **`Program result`**.

Note:

1. The value of the argument `tobank` is a valid `Bank` object, so you don't need to check.
2. The values of `toaccid` and `money` arguments are positive integers.
3. You must not use `self.balance` in the method. If you use it, you will get zero point.
4. You have to use string formatting operators to print the result.
   Do not concatenate strings.

(5-4)	