MASSEY UNIVERSITY
MANAWATU AND ALBANY CAMPUSES

EXAMINATION FOR

159.171 COMPUTATIONAL THINKING AND SOFTWARE DEVELOPMENT

Semester One – 2014

Time allowed: **TWO (2)** hours.

CALCULATORS ARE NOT PERMITTED

Answer ALL the questions.

An appendix containing Python functions and methods is attached.

This exam is to be written on and submitted for marking.
Please provide student ID and name details below.

Student ID:	1	0	1	5	8	9	3	1

Family Name:	MCDONALD
First Name/s:	BRETT LEE DAVID

Question 1 **[5 marks]**

(a) For each of the following three Python statements, show what will be output or if it gives an error, explain why it is wrong.

```
print 37 + 37
print 37 + "37"
print "37" + "37"
```

```
# Question 1a
# answer
#74
#error - can't add types int and string
#3737
# for testing:
print(37 + 37)
print ('37' + '37')
print (37 + '37')
```

(b) It is often necessary to display the values of several variables on a single line. For the following Python assignment statements:

```
name = "Sue"
age  = 20
height = 1.7214  # metres
```

Show how you can display the values *from these variables* on a single line so the result is:

```
Sue is 20 and is 1.72m tall
```

Note that the height has been rounded to two decimal places.

```
# Question 1b
# for testing:
name = 'Sue'
age = 20
height = 1.7214
# answer:
print('{} is {} and is {:.2f}m tall'.format(name,age,height))
```

Question 2 **[10 marks]**

(a) **Using a *while* loop**, write a Python program that calculates and then displays the numbers 1, 8, 15 … (going up in steps of 7). The program should stop when the calculated value exceeds 250 without printing it.

```
# Question 2a
# answer:
count = 1
while count < 251:
    print(count)
    count += 7
```

(b) **Write a modified version of the *while* loop from part (a)** so when the value is between *100-105* or between *200-205*, the string *"Excluded"* is displayed instead of the number.

```
# Question 2b
# answer:
count = 1
while count < 251:
    if 100 <= count <= 105 or 200 <= count <= 205:
        print('Excluded')
    else:
        print(count)
    count += 7
```

Question 3 **[8 marks]**

There are five variables that relate to a patient's condition:

1. *temperature* (a float, we will class "normal" as 36 to 38, and above 38 as "high")
2. *spots* (True/False)
3. *blockedNose* (True/False)
4. *wateryEyes* (True/False)
5. *season* (a string – either ‘winter’, ‘summer’, ‘spring’ or ‘autumn’)

Write a sequence of statements that will set the variable *diagnosis* to either *"hay fever"*, *"flu"*, *"measles" or “healthy”* according to the following rules:

- give a diagnosis of *"hay fever"* if a patient has temperature in the normal range, has a blocked nose or watery eyes, and the season is *spring* or *summer*.
- if they have spots and a high temperature, then diagnose *"measles"*
- if they have a blocked nose and a high temperature, diagnose *"flu"*
- otherwise set *diagnosis* to *“healthy”*

```
# Question 3
# for testing:
temperature = 39
spots = False
blockedNose = True
wateryEyes = False
season = 'winter'
# answer:
t = temperature
s = spots
b = blockedNose
w = wateryEyes
s = season
if 36 <= t <= 38 and (b or w) and (s == 'spring' or s == 'summer'):
    diagnosis = 'hay fever'
elif s and t > 38:
    diagnosis = 'measles'
elif b and t > 38:
    diagnosis = 'flu'
else:
    diagnosis = 'healthy'
# for testing:
print(diagnosis) # should be 'flu'
```

Question 4 **[10 marks]**

(a) **Write a function *getReply(msg, validReplies)***. The parameter *msg* is a string, and *validReplies* is a list of strings.

The function displays *msg* and waits for the user to type a reply. If the reply is one of the strings in *validReplies*, **the function returns the reply**, otherwise the user is prompted repeatedly until a valid response is given.

```
# Question 4a
# answer:
def getReply(msg, validReplies):
    while True:
        response = input(msg)
        if response in validReplies:
            return response
# for testing:
msg = 'yes or no'
validReplies = ['yes','no']
print(getReply(msg,validReplies))
```

(b) **Write a version of *getReply*() from part (a) that has a third parameter *maxTries* (an integer)**. In this version of the function, if the user hasn't given a valid reply after *maxTries* attempts, the function returns with *None* instead of the user's reply.

Show how you would call this function and then test the returned value to display either the reply or, if the user didn't give a valid reply, display *"No valid reply"*.

```
# Question 4b
# answer:
def getReplyV2(msg, validReplies, maxTries):
    count = 0
    while count < maxTries:
        response = input(msg)
        if response in validReplies:
            return response
        count += 1
    return

returned_value = getReplyV2(msg='yes or no: ', validReplies = ['yes','no'], maxTries = 5)
if returned_value:
    print(returned_value)
else:
    print('No valid reply')
# for testing: run the code above with 'x' 5 times, it should print 'no valid reply',
# then run it with 'yes', it should print 'yes'
```

Question 5 **[8 marks]**

Use a *for* or *while* loop (or loops) in your answer to the following:

(a) **write a function *countWords()*** that

- **has two parameters**, a target word (*thisWord*) and a list of words (*listOfWords*).
- **returns a count** of the number of times that *thisWord* occurs in the list of words.

```
# Question 5a
# answer:
def countWords(thisWord, listOfWords):
    count = 0
    for word in listOfWords:
        if word == thisWord:
            count += 1
    return count
# for testing:
print(countWords(thisWord='cat', listOfWords=['cat','cats','dog','dogs'])) # should print '1'
```

(b) **Write a more general *countWordS()* function** that gives a count of how many times each word in a list of words *(targetWords*) occurs in *listofWords*. It prints out a count of how many times each word occurs.

An example of its use might look like this:

```
wordlist = ['big', 'cats', 'like', 'really', 'really', 'really', 'big', 'fish']
countWordS(['frog', 'really', 'fish', 'big'], wordlist )
```

and it displays something like:

```
"frog" occurs 0 times.
"really" occurs 3 times.
"fish" occurs 1 times.
"big" occurs 2 times.
```

***Hint:* this task is simpler if you use the function *countWords()* from part (a).**

```
# Question 5b
# answer:
def countWordsV2(targetWords, listofWords):
    for word in targetWords:
        print('"{}" occurs {} times.'.format(word,countWords(word, listofWords)))
# for testing:
def countWords(thisWord, listOfWords):
    count = 0
    for word in listOfWords:
        if word == thisWord:
            count += 1
    return count
countWordsV2(targetWords=['frog','really','fish','big'], listofWords =
['big','cats','like','really','really','really','big','fish'])
# should display:
# "frog' occurs 0 times.
# "really" occurs 3 times.
# "fish" occurs 1 times.
# "big" occurs 2 times.
```

Question 6 **[10 marks]**

(a) What is the decimal value of the hexadecimal number ***A5*** ?

What is the decimal value of the binary number ***0110010*** *?*

```
# Question 6a
# answer:
165
50
# checked online, these values are correct
```

(b) Using *raw_input(),* ***write a short program*** that enables a user to enter a floating number. Then the program either prints the numeric value of the entered number, or if the user inputs something that cannot be converted into a number, displays *"Invalid Number"*.

```
# Question 6b
# answer:
r = input('Enter a floating number: ')
try:
    r = float(r)
    print(r)
except:
    print('Invalid Number')
# for testing: run the above code with 'x' - should print 'Invalid Number',
# and with '9', should print 9.0, and with 9.56, should print 9.56
```

(c) **Write a program** that reads lines from the file ***in.txt*** and writes these lines to a new file ***out.txt***. Copy all lines to the *out.txt* file except: empty lines, and lines that starts with a **#**.

```
# Question 6c
# for testing:
with open('in.txt','w') as intxt:
    intxt.writelines(['line 1\n','\n',' # line 3\n', 'line 4\n'])
# answer:
with open('in.txt','r') as intxt, open('out.txt','w') as outtxt:
    for line in intxt:
        if line != "\n" and line.lstrip()[0] != '#':
            outtxt.write(line)
# for testing: read the file 'out.txt', it should only have two lines: 'line 1' and 'line 4'
```

Question 7 **[10 marks]**

Here is a partial declaration of a class *Room* that has two float attributes *length* and *width*.

```
class Room:
  def __init__(self):
    self.length = 1.0
    self.width = 1.0

# set the length & width of the room object to newL, newW respectively
  def setDim(self, newL, newW):
    # missing code

# returns the area of the room object as a float value
  def area(self):
    # missing code
```

(a) **Write the code for the two methods *setDim* and *area***. Assume the room is rectangular so its area is the product of its dimensions.

(b) **Write a Python program** to

i. create a *room* object
ii. display the area of this object
iii. change the dimensions of the *room* object so its length is 15 and its width is 10
iv. display the area of the object

Question 8 **[4 marks]**

(a) **Write a single Python statement** so that the variable *fruitInStock* is changed from:

```
fruitInStock  = {'apples':1, 'oranges':2, 'pears':3}
```

to

```
fruitInStock  = {'apples':1, 'oranges':2, 'pears':3, 'bananas':4}
```

Note: The order of items in *fruitInStock* is not important.

```
# Question 8a
# for testing:
fruitInStock = {'apples':1, 'oranges':2, 'pears':3}
# answer:
fruitInStock['bananas'] = 4
# for testing:
print(fruitInStock)
```

(b) **Write a Python statement (or two) to** find the value associated with the key 'pears' and update the value associated with key 'apples' to be the same.

```
fruitInStock  = {'apples':1, 'oranges':2, 'pears':3, 'bananas':4}
```

```
# Question 8b
# for testing:
fruitInStock = {'apples':1, 'oranges':2, 'pears':3, 'bananas':4}
# answer:
fruitInStock['apples'] = fruitInStock['pears']
# for testing:
print(fruitInStock)
```

(c) How can you test if the dictionary contains a certain key? **A textual (written) explanation is sufficient**, although you can write Python code if you prefer.

```
# Question 8c
# answer:
# print(key in dict) # this will print True if the key is in the dictionary, otherwise it will print False.
# for testing:
fruitInStock = {'apples':1, 'oranges':2, 'pears':3, 'bananas':4}
print('apples' in fruitInStock) # should print True
print('apple' in fruitInStock) # should print False
```

Question 9 **[10 marks]**

(a) A comma-separated value (CSV) text file is often used to export data from spreadsheets. **Write a program that reads lines from a CSV file and builds a list of lists**. Each sub-list within the main list contains the fields from a single line in the file. Where possible, fields are converted into a number. Strings have leading/trailing whitespace removed.

A CSV file containing contains three values values per line (e.g. Id, Name, Balance) e.g.

```
1010,Bill,145.98
1147,Gina,10288.97
2917,  Willie, 4.97
```

would result in a list like this:

```
[ [1010,'Bill',145.98], [1147,'Gina',10288.97], [2917,'Willie',4.97] ]
```

(b) **Show how you could create a dictionary called *entries*** so that ***`entries["Gina"]`*** would return the sublist `[1147,'Gina',10288.97]`. You can assume that the list created in part (a) is available in a variable called `entryList`.

If you wish, you can create the dictionary for part (b) while building the list for part (a), rather than building it later. This will combine parts (a) and (b) into a single program, which is fine.

```
# Question 9a and 9b
for testing:
with open('CSV.txt','w') as csv:
   csv.write('1010,Bill,145.98\n1147,Gina,10288.97\n,2917, Willie, 4.97')
# answer:
entries = {}
listOfLists = []
with open('CSV.txt','r') as csv:
   for line in csv:
      subList = []
      for item in line.split(','):
         if item.strip().isdigit():
            subList.append(int(item.strip()))
         else:
            try:
               subList.append(float(item.strip()))
            except:
               subList.append(item.strip())
      if subList[1] not in entries:
         entries[subList[1]] = subList
      listOfLists.append(subList)
# for testing:
print(entries) # should print entries for Bill, Gina and Willie
print(entries['Gina']) # should print entry for Gina
print(listOfLists) # should print listOfLists
```

+ + + + + + + +

Reference List of Useful Python functions and methods	
Functions:	
raw_input([prompt]) → str	Read a string from standard input.
abs(x) → number	Return the absolute value of x.
chr(x) → str	Returns the string value of x
ord(x) → int	Returns the ASCII value of x
int(x) → int	Convert x to an integer, if possible.
float(x) → float	Convert x to a float value, if possible.
len(x) → int	Return the length of (string, tuple or list or dictionary) x
max(iterable) → object	With a single iterable argument, return its largest item.
max(a, b, c, ...) → object	With two or more arguments, return the largest argument.
min(iterable) → object	With a single iterable argument, return its smallest item.
min(a, b, c, ...) → object	With two or more arguments, return the smallest argument.
open(name[, mode]) → file open for reading, writing	Open a text file. Legal modes are "r","rt" (read), "w", "wt" (write)
range([start], stop, [step]) → list-like-object of int	Return the integers starting with *start* and ending with *stop-1* with *step* specifying the amount to increment (or decrement).
dict:	
x in D → bool	Returns True if x is a key in D
D[k] → object	Produce the value associated with the key k in D.
del D[k]	Remove D[k] from D.
D.clear()	Sets D to empty dictionary
D.copy()	Returns a copy of D
D.get(k) → object	Return D[k] if k in D, otherwise return None.
D.keys() → list-like-object of object	Return the keys of D.
D.values() → list-like-object of object	Return the values associated with the keys of D.
D.items() → list-like-object of (object, object)	Return the (key, value) pairs of D, as 2-tuples.
files	
F.close() → NoneType	Close the file.
F.read() → str	Read until EOF (End Of File) is reached, and return as a string.
F.readline() → str	Read and return the next line from the file, as a string. Retain newline. Return an empty string at EOF.
F.readlines() → list of str	Return a list of the lines from the file. Each string ends in a newline.
F.writeline(s)	Write the string s to the file.

F.writelines(list of str)	Write a sequence of strings to the file. writelines() does **not** add line separators.
list:	
x in L → bool	Produce True if x is in L and False otherwise.
L.append(x) → NoneType	Append x to the end of the list L.
L.count(x)	Returns the number of occurrences of x in L
L.insert(index, x) → NoneType	Remove the first occurrence of value from L.
L.remove(value) → NoneType	Reverse *IN PLACE*.
L.reverse() → NoneType	Sort the list in ascending order.
L.sort() → NoneType	
str:	
x in S → bool	Produce True if and only if x is in S.
str(x) → str	Convert an object into its string representation, if possible.
S.capitalize() → str	Return a copy of the string S, capitalised.
S.endswith(suffix)	Return True if the string ends with the specified suffix, otherwise return False
S.find(sub[, i]) → int	Return the lowest index in S (starting at S[i], if i is given) where the string sub is found or -1 if sub does not occur in S.
S.isdigit() → bool	Return True if all characters in S are digits and False otherwise.
S.islower()	Return True if all characters in S are lower case and False otherwise.
S.isupper()	Return True if all characters in S are uppercase and False otherwise.
S.lower() → str	Return a copy of the string S converted to lowercase.
S.replace(old, new) → str	Return a copy of string S with all occurrences of the string old replaced with the string new.
S.split([sep]) → list of str	Return a list of the words in S, using string sep as the separator and any whitespace string if sep is not specified.
S.startswith(prefix)	Return True if string starts with the prefix, otherwise return False.
S.strip() → str	Return a copy of S with leading and trailing whitespace removed.
S.title() → str	Return a copy of the string S with the first letter of each word capitalised.
S.upper() → str	Return a copy of the string S converted to uppercase.