1501/159.171 ASR
MAN/ALB 2Hr
Distance/Internal NSB

MASSEY UNIVERSITY
MANAWATU AND ALBANY CAMPUSES

EXAMINATION FOR

159.171 COMPUTATIONAL THINKING AND SOFTWARE DEVELOPMENT

Semester One – 2015

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

EXAMINER TO COMPLETE THIS TABLE		
Questions	**Total Marks of Questions**	**Actual Marks Obtained**
1	12	
2	17	
3	7	
4	18	
5	10	
6	16	
Total	**80**	

Question 1 [Total: 12 marks]

(a) For each of the following three Python statements, show what will be displayed.

```
print ("37" + "37")
print ("Hello"[1])
print ('37'.replace('3', '7'))
```

```
# Question 1a
# answer:
#3737
#e
#77

# for testing:
print ("37" + "37")
print ("Hello"[1])
print('37'.replace('3', '7'))
```

[3 marks]

(b) Translate the following equation into Python, where the LC term indicates that L is multiplied by C

$$f = \frac{1}{2\pi\sqrt{(LC)}}$$

[3 marks]

```
# Question 1b
# for testing:
L = 2
C = 4
# answer:
import math

f = 1 / (2 * math.pi * (L*C)**0.5)

# for testing:
print(f)
```

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

```
PetrolGrade = "Super"
Amount = 25             # Litres
costPerLitre = 1.75     # Dollars
```

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

```
The cost of 25 litres of Super is $43.75
```

Write your code so that the output will always be displayed rounded to **two decimal places**.

[3 marks]

```
# Question 1c
# for testing:
PetrolGrade = 'Super'
Amount = 25
costPerLitre = 1.75

# answer:
print('The cost of {} litres of {} is
${:.2f}'.format(Amount,PetrolGrade,Amount*costPerLitre))
```

(d) **What is the output of the following code?**

```
s = '3'
n = 3

print(s*3)
print(n*3)
print(int(s)*3)
print(str(n)*3)
```

[3 marks]

```
# Question 1d
# answer:
#333
#9
#9
#333
# for testing:
s = '3'
n = 3
print(s*3)
print(n*3)
print(int(s)*3)
print(str(n)*3)
```

Question 2 **[Total: 17 marks]**

(a) **Using a *for* loop and the *range* function**, write a Python program that creates a list of all the numbers from 1 to 25 in steps of 3. Print the list after it has been created; the output should look like this:

```
[ 1, 4, 7, 10, 13, 16, 19, 22, 25 ]
```

[3 marks]

```
# Question 2a
# answer:
list = []
for i in range(1,26,3):
   list.append(i)
print(list)
# for testing:
# should display [1,4,7,10,13,16,19,22,25]
```

(b) **Using values from the user and a loop**, write a programme to generate the output shown).

For those values that are between *lower* and *upper* (inclusive), display the number and "BETWEEN", otherwise display the number and *"Excluded"*.

The output should look like this:

```
Stepsize   ? 3.5
Lower limit? 12
Upper limit? 20

1.5 Excluded
5.0 Excluded
8.5 Excluded
12.0 BETWEEN
15.5 BETWEEN
19.0 BETWEEN
22.5 Excluded
```

[5 marks]

```
# Question 2b
# answer:
step = float(input('Stepsize   ?'))
low = float(input('Lower limit?'))
up = float(input('Upper limit?'))
start = 1.5
end = 23
while start < end:
    if low <= start <= up:
        print('{:.1f} BETWEEN'.format(start))
    else:
        print('{:.1f} Excluded'.format(start))
    start += step
```

(c) **Using a *while* loop**, write a Python program that displays each of the characters in "Hello" on a new line, with the letter number before each character, starting at 1.

e.g. 1: H
2: e ***etc.***

[5 marks]

```
# Question 2c
# answer:
count = 1
string = 'Hello'
while count < 6:
    print('{}: {}'.format(count,string[count-1]))
    count += 1
```

(d) What is the output of the following program?

```
for numA in range(2):
    print(1)
    for numB in range(3):
        print(2, end = " ")
    print()
```

[4 marks]

```
# Question 2d
# answer:
#1
#2 2 2
#1
#2 2 2

# for testing:
for numA in range(2):
    print(1)
    for numB in range(3):
        print(2, end = " ")
    print()
```

Question 3 **[Total: 7 marks]**

(a) There are five variables (with the noted values) that relate to a patient's condition:

1.	*Temperature*	which is either *Normal (<=37)* or *High (i.e. > 37)*
2.	*hadFluShot*	*(True/False)*
3.	*PollenAllergy*	*(True/False)*
4.	*wateryEyes*	*(True/False)*
5.	*Season*	*('Spring', 'Summer', 'Autumn', 'Winter')*

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

- give a ***diagnosis*** of *"hay fever"* if a patient has a normal temperature, has watery eyes, and has a PollenAllergy
- if it's Winter, they have a high temperature and have NOT had a FluShot, ***diagnose*** *"flu"*
- if they have a high temperature, but it's not Winter, ***diagnose*** *"a virus"*
- otherwise set ***diagnosis*** to *"healthy"*

[5 marks]

```
# Question 3a
# for testing:
Temperature = 38
hadFluShot = True
PollenAllergy = True
wateryEyes = True
Season = 'Spring'
# answer:
t = Temperature
h = hadFluShot
p = PollenAllergy
w = wateryEyes
s = Season
if t <= 37 and w and p:
    diagnosis = 'hay fever'
elif s == 'Winter' and t > 37 and not h:
    diagnosis = 'flu'
elif t > 37 and not s == 'Winter':
    diagnosis = 'a virus'
else:
    diagnosis = 'healthy'
# for testing:
print(diagnosis) # should be 'a virus'
```

(b) What is the output of the following code?

```
def multiply(number):
    p = number * 2
    return p

def testNumber(number):
    result = 0
    if(number >= 7 and number < 20):
        result = multiply(number)
    return result


first_result = testNumber(5)
print(first_result)

second_result = testNumber(10)
print(second_result)
```

[2 marks]

```
# Question 3b
# answer:
#0
#20
# for testing:
def multiply(number):
    p = number * 2
    return p

def testNumber(number):
    result = 0
    if(number >= 7 and number < 20):
        result = multiply(number)
    return result

first_result = testNumber(5)
print(first_result)

second_result = testNumber(10)
print(second_result)
```

Question 4 [Total: 18 marks]

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

The function displays *msg* and waits for the user to type a reply. Any reply is acceptable (and returned) EXCEPT for those in *bannedReplies*. If the reply is one of the strings in *bannedReplies*, the user is prompted repeatedly until a valid (i.e. non-banned) response is given.

The test for a banned replies is case-insensitive, so if *bannedReplies* was *['cheeeze', 'toast']* and the user entered TOAST, then this reply would *not* be accepted.

[7 marks]

```
# Question 4a
# answer:
def getReply(msg, bannedReplies):
    while True:
        response = input(msg)
        if response.lower() not in [x.lower() for x in bannedReplies]:
            return response
# for testing:
msg = 'reply: '
bannedReplies = ['yes','no']
print(getReply(msg,bannedReplies))
```

(b) **Write a version of *getReplyV2*(msg, bannedWordlist, bannedWordListFilename**) that behaves in the same way as part(a) but has a third parameter that is the filename of a list of banned words. The words in the file are stored one per line.

If the file can be opened, the words in the file are added to those in the *bannedWordlist* parameter. If the file doesn't exist, no error occurs and the behaviour is the same as part (a). It's acceptable to make use of the *getReply()* function you created in from part (a) .

[6 marks]

```
# Question 4b
# answer:
def getReplyV2(msg, bannedWordlist, bannedWordListFilename=False):
    if bannedWordListFilename:
        with open(bannedWordListFilename,'r') as f:
           for line in f:
               bannedWordlist.append(line)
    response = getReply(msg, bannedWordlist)
    return response
# for testing:
msg = 'reply: '
bannedWordList = ['yes','no']
with open('bannedWords.txt','w') as f:
    f.write('maybe')
print(getReplyV2(msg, bannedWordList)) # should accept 'maybe'
bannedWordListFilename = 'bannedWords.txt'
print(getReplyV2(msg, bannedWordList, bannedWordListFilename)) # should not accept
'maybe'
```

Question 4 continued

(c) **Write a program that can take a list of words and calculate how many times each vowel occurs:**

e.g. for

```
wordlist = ['big', 'cats', 'like', 'really']
```

it might display something like:

```
"a" occurs 2 times.
"e" occurs 2 times.
"i" occurs 2 times.
"o" occurs 0 times.
"u" occurs 0 times.
```

You can (but it not necessary to) use lists/dictionaries to answer this. The simpler option of using multiple variables is acceptable.

[5 marks]

```
# Question 4c
# for testing:
wordlist = ['big','cats','like','really']
# answer:
dict = {'a': 0, 'e': 0, 'i': 0, 'o': 0, 'u': 0}
for word in wordlist:
    for ch in word:
        if ch.lower() in dict:
            dict[ch.lower()] += 1
for key in dict:
    print('"{}" occurs {} times.'.format(key,dict[key]))

# or...

vowels = [['a', 0],['e',0],['i',0],['o',0],['u',0]]
for word in wordlist:
    for ch in word:
        for vowel in vowels:
            if ch.lower() == vowel[0]:
                vowel[1] += 1
for vowel in vowels:
    print('"{}" occurs {} times.'.format(vowel[0],vowel[1]))
```

Question 5 **[Total: 10 marks]**

(a) Using *input()*, **write a function with one parameter, a prompt that is displayed to the user.** The function allows the users to enter a floating point number and then returns this number. If the user inputs something that cannot be converted into a number, *"Invalid Number"* is displayed.

The function does not return until a valid number has been entered OR the user presses the *Enter* key without entering any other characters, in which case the function returns the value *None*.

[5 marks]

```
# Question 5a
# answer:
def float_check(msg):
    while True:
        response = input(msg)
        if response == "":
            return
        else:
            try:
                return float(response)
            except:
                print('Invalid Number')
# for testing:
msg = 'enter num: '
print(float_check(msg))
```

(b) Write a program that reads lines from the file *numbers.txt*, where each line contains two numbers. Your program should calculate the product of the numbers on each line (i.e. multiply them) and then calculate the sum of these products.

e.g. for a file containing

```
3    2
20   1
5   100
```

Your program would display something like

```
There were 3 lines
The total of the 3 lines is 526
```

[5 marks]

```
# Question 5b
# for testing:
with open('numbers.txt','w') as f:
    f.write('3 2\n')
    f.write('20 1\n')
    f.write('5 100\n')
# answer:
lines = 0
sum = 0
with open('numbers.txt','r') as f:
    for line in f:
        lines += 1
        sum += (int(line.strip().split()[0]) * int(line.strip().split()[1]))
print('There were {} lines'.format(lines))
print('The total of the {} lines is {}'.format(lines,sum))
```

Question 6 [Total: 16 marks]

For Questions 6 (a) and 6 (b), two functions *createStats()* and *wordReport()* that enable you to take a multi-line string and list the words it contains, how many times each word occurs and the line on which a word first appears, and then display a report.

e.g. For

```
s = """mary had
a little lamb with a
blue fleece and
it had a little blue eye"""
```

It might produce:

```
Word          a   Count: 3  First Line 2
Word        and   Count: 1  First Line 3
Word       blue   Count: 2  First Line 3
Word        eye   Count: 1  First Line 4
Word     fleece   Count: 1  First Line 3
Word        had   Count: 2  First Line 1
Word         it   Count: 1  First Line 4
Word       lamb   Count: 1  First Line 2
Word     little   Count: 2  First Line 2
Word       mary   Count: 1  First Line 1
Word       with   Count: 1  First Line 2
```

(a) **Write the function *createStats(s)*** that takes a multi-line string (s) and builds a dictionary indexed by word, that contains the count and first line information.

[8 marks]

```
# Question 6a
# answer:
def createStats(s):
   dict = {}
   sList = s.split('\n')
   for line in sList:
      for word in line.split():
         if word not in dict:
            dict[word] = [1,0]
            dict[word][1] = sList.index(line) + 1
         else:
            dict[word][0] += 1
   return dict
# for testing:
s = """mary had
a little lamb with a
blue fleece and
it had a little blue eye"""
print(createStats(s))
```

(b) **Write the function wordReport()** that displays the report, using the dictionary created in part (a).

[4 marks]

```
# Question 6b
# answer:
def wordReport(s):
    dict = createStats(s)
    for key in dict:
        print('Word {:>10} Count: {} First Line {}'.format(key,dict[key][0],dict[key][1]))
# for testing:
s = """mary had
a little lamb with a
blue fleece and
it had a little blue eye"""
wordReport(s)
```

(c) Data on your MP3 collection has been stored in a nested Python list called *MP3List*. This list contains a sub-list for each song, where the sub-list contains the artist's name, the track name, and the album name.

Write a function that takes two parameters the *MP3List*, and a *word*. The function should then display the details of all tracks that contain the word somewhere in the track name. The displayed output should be formatted so the artist's name, track name and album name appear in columns.

[4 marks]

```
# Question 6c
# answer:
def find_track(MP3List,word):
    for entry in MP3List:
        if word in entry[1]:
            print('Artist: {:>10}, Track: {:>10}, Album: {:>10}'.format(entry[0],entry[1],entry[2]))
# for testing:
MP3List = [['kanye west','monster','dark fantasy'],['BEP','my humps','monkey biz']]
word = 'mon'
find_track(MP3List,word)
find_track(MP3List,word='my')
```

+ + + + + + + +

Please **ensure that you have completed your student details correctly** where indicated in the fields provided on the **front page** of this examination paper

Reference List of Useful Python functions and methods	
Functions:	
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.