MASSEY UNIVERSITY
MANAWATU, ALBANY, HEBEI UNIVERSITY OF TECHNOLOGY AND DISTANCE CAMPUSES

EXAMINATION FOR

159.171 COMPUTATIONAL THINKING AND SOFTWARE DEVELOPMENT

Semester One – 2017

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

CALCULATORS ARE NOT PERMITTED

Answer ALL the questions.

An appendix containing Python functions and methods is attached.

Write on this exam and submit it for marking.

Massey Student ID:	1	0	1	5	8	9	3	1

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

EXAMINER TO COMPLETE THIS TABLE		
Question	**Total Marks of Questions**	**Actual Marks Obtained**
1	12	
2	16	
3	7	
4	18	
5	16	
6	11	
Total	**80**	

Question 1 **[12 marks]**

(a) For each of the following Python statements, what will be displayed?

```
print(int(1234.56))
print("Nice Day".upper().split('A'))
print(len('55.3')*10)
print(15 % 2 == 0)
print(">>%0.3f<<" % (3.1415926))
```

[5 marks]

```
# Question 1a
# Learnings: int() doesn't apply rounding to the decimal point, it just trims it off.
# answer:
# 1234
# ['NICE D','Y']
# 40
# False
# >>3.142<<
# for testing:
print(int(1234.56))
print("Nice Day".upper().split('A'))
print(len('55.3')*10)
print(15 % 2 == 0)
print(">>%0.3f<<" % (3.1415926))
```

(b) Translate the following equation into Python, where the two terms being adjacent indicates multiplication (e.g. ***7x*** indicates that **7** is multiplied by ***x***). Assume that the variable x has already been defined.

$$y = \sqrt{3x^3 - 5x^2 + 7x + 9}$$

[2 marks]

```
# Question 1b
# for testing:
x = 2
# answer:
y =  ((3*x**3) - (5*x**2) + (7*x) + 9)**0.5
# for testing:
print(y) # should be about 5
```

Question 1 continued

(c) Using a loop and data from the following lists:

```
title  = ['roof', 'walls', 'doors']
colors = ['green', 'blue', 'red']
```

write some code that will print *green roof*, *blue walls*, *red doors*, each on a separate line.

[3 marks]

```
# Question 1c
# for testing:
title = ['roof', 'walls', 'doors']
colors = ['green', 'blue', 'red']
# answer:
for i in range(len(colors)):
    print('{} {}'.format(colors[i], title[i]))
# for testing: it should display green roof, blue walls, red doors on separate lines.
```

(d) The following code correctly calculates the volume of a cylinder:

```
import math
def volume(radius, length): # calculate the volume of a cylinder
     print(int(math.pi * radius**2 * length))

print("Volume =", volume(2, 10))
```

but does not (as intended) display

```
Volume = 125
```

Instead it displays

```
125
Volume = None
```

Explain why *None* is printed and how you would fix the error.

[2 marks]

```
# Question 1d
# answer:
#'None' is printed because def volume() does not have a return statement. To fix the
error, within def volume(), replace print() with return.
# for testing:
import math
def volume(radius, length):
    return int(math.pi * radius**2 * length)
print("Volume =", volume(2,10)) # should display 'Volume = 125'
```

Question 2 **[16 marks]**

(a) Write a function ***update(L, word, word2)*** that will replace the first occurrence of ***word*** in the list ***L*** with ***word2***.If the replacement was successful, return both the list **L** and **True**. If the word is not in the list, return the list **L** and **False**

e.g. `update( ["hello", "hi", "welcome", "bye"] , "welcome", "goodbye")`

would return `["hello", "hi", "goodbye", "bye"]` and `True`

[4 marks]

```
# Question 2a
# Learnings: Make sure not say 'else' when you mean 'except' in a try/except block.
# answer:
def update(L, word, word2):
    try:
        L[L.index(word)] = word2
        return L, True
    except:
        return L, False
# for testing:
print(update(['hello', 'hi', 'welcome', 'bye'] , 'welcome', 'goodbye'))
# it should display ['hello', 'hi', 'goodbye', 'bye'] and True
```

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

```
for colour in ['red', 'green', 'blue']:
   for value in range(1, 5, 2):
         print(value*colour)
```

[4 marks]

```
# Question 2b
# answer:
# red
# redredred
# green
# greengreengreen
# blue
# blueblueblue
# for testing:
for colour in ['red', 'green', 'blue']:
    for value in range(1,5,2):
        print(value*colour)
```

Question 2 continued

(c) **Write a program** that takes a list of strings **L** and builds a new list of sublists **K**. The first element in each sublist is the length of the item from L, the second is the item itself.

e.g if the input list L was:

```
L = [ 'It is', 'time', 'for', 'tea' ]
```

The output list K would contain:

```
[ [5,'It is'], [4,'time'], [3,'for'], [3,'tea'] ]
```

[4 marks]

```
# Question 2c
# for testing:
L = ['It is', 'time', 'for', 'tea']
# answer:
K = []
for item in L:
    subList = []
    subList.append(len(item))
    subList.append(item)
    K.append(subList)
# for testing:
print(K) # it should be [[5,'It is'], [4,'time'],[3,'for'],[3,'tea']]
```

(d) **Using a *while* loop**, write a Python program that displays all values from 1 to 30, except those values that are multiples of 5 or multiples of 7.

[4 marks]

```
# Question 2d
# Learnings: when using % for multiple checking, make sure to compare it to 0.
# answer:
count = 1
while count < 31:
    if count % 5 != 0 and count % 7 != 0:
        print(count)
    count += 1
# for testing: It shouldn't display multiples of 5 or 7.
```

Question 3 **[7 marks]**

(a) For the following variables:

```
location   : 'inside', 'outside'
weather    : 'sunny', 'cloudy',  'rainy', 'stormy'
time       : 0000-2359   # 0000 = midnight, 2359 = 11:59pm
wind_speed : 0-100km/h
```

set the variables, *use_umbrella* and *need_sunglasses* to **True** if the following conditions hold, and **False** otherwise:

- **use_umbrella** if all the following are true: you're outside, the weather is rainy or stormy, and the wind speed is <30km/h
- **need_sunglasses** if you're outside, it's between 10am and 4pm and it's sunny

You can use if-statements or any Python constructs you think appropriate.

[4 marks]

```
# Question 3a
# Learnings: make sure to use == for comparison, and not =.
# for testing
location = 'outside'
weather = 'sunny'
time = 1200
wind_speed = 50
# answer:
l = location
w = weather
t = time
wind = wind_speed
use_umbrella = False
need_sunglasses = False
if l == 'outside' and (w == 'rainy' or w == 'stormy') and wind < 30:
    use_umbrella = True
if l == 'outside' and 1000 <= t <= 1600 and w == 'sunny':
    need_sunglasses = True
# for testing:
print(use_umbrella) # should be False
print(need_sunglasses) # should be True
```

Question 3 continued

(b) What if anything, is printed by the following block of code:

```
def f2(x, y):
    if not x > y:
        return "Red"
    else:
        return "Blue"

def Test(x):
    if x < 0:
        p = f2(2,4)
    else:
        p = f2(4,2)

    print(x**2, p)

Test(5)
```

[3 marks]

```
# Question 3b
# answer:
# 25 blue
# for testing:
def f2(x, y):
    if not x > y:
        return 'Red'
    else:
       return 'Blue'
def Test(x):
    if x < 0:
        p = f2(2,4)
    else:
        p = f2(4,2)
    print(x**2,p)
Test(5)
```

Question 4 **[18 marks]**

(a) Write a program that reads the contents of a file named *input.py* and writes the file *output.py*, omitting empty lines or lines that start with a # character.

e.g. if the file *input.py* contains:

```
# This is a file of categories
Animals: cat, dog, frog, cheetah, tiger
Items: desk, chair, bus, cups, pencil

Technology: cellphone, TV, laptop, wifi-router
    # That's end of the file
```

The output would omit lines 1, 4 and 6, so the file *output.py* would contain:

```
Animals: cat, dog, frog, cheetah, tiger
Items: desk, chair, bus, cups, pencil
Technology: cellphone, TV, laptop, wifi-router
```

[6 marks]

```
# Question 4a
# Learnings: the line.lstrip()[0] operation will cause an 'index out of range' error on empty lines. Therefore, always use a try/except block when checking for empty lines and '#' ad the same time.
# for testing:
with open('input.py','w') as input:
    input.write("# This is a file of categories\nAnimals: cat, dog, frog, cheetah, tiger\nItems: desk, chair, bus cups, pencil\n\nTechnology: cellphone, TV, laptop, wifi-router\n\t\t# That's end of the file")
# answer:
with open('input.py','r') as input, open('output.py','w') as output:
    for line in input:
        try:
            if line.lstrip()[0] != '#' and line != '\n':
                output.write(line)
        except:
            pass
# for testing:  The output should be lines 1, 4 and 6 (start with Animals, Items and Technology)
```

Question 4 continued

b1) A simple number-to-text translator can be based on a list of pairs:

```
pairs = [ [0, 'zero'],  [1, 'one'],  [2, 'two'],  [3, 'three'],  [4, 'four'],  [5, 'five'] ]
```

Write a function called **lookup()** that takes a single number (an int) as a parameter and **returns** (**but does NOT print)** the textual version of that number.

b2) Write code that asks the user for a number (**x**) between 1 and 5. Then for the values 1-5, it calls your function **lookup().** The value and its textual version are displayed *except for the value (**x**) chosen by the user:*

For **x**, you can assume the user enters a valid number and it will be between 1 and 5.
e.g.

```
Omit which number? 4
1  -->  one
2  -->  two
3  -->  three
5  -->  five
```

[6 marks]

```
# Question 4b1
# Learnings: always double-check bracket balancing, especially brackets are nested several times.
# answer:
def lookup(int_num):
    int_dict = {0: 'zero', 1: 'one', 2: 'two', 3: 'three', 4: 'four', 5: 'five'}
    return int_dict[int_num]
# for testing:
print(lookup(4)) # it should display 'four'
# Question 4b2
# answer:
int_list = [1,2,3,4,5]
remove_num = int(input('Omit which number? '))
int_list.pop(int_list.index(remove_num))
for i in int_list:
    print('{} --> {}'.format(i,lookup(i)))
# for testing: It should show everything except 4 if 4 is given as the argument.
```

Question 4 continued

(c) The following function should display a question and then let the user enter a reply. It then checks the answer and only returns the reply if it is one of the valid answers.

The user input is case-independent, so 'Y', 'YES' and 'yes' are all acceptable but the returned reply should always be in lowercase.

Unfortunately the function contains several errors.

```
def getReply(question, valid_answers):
    #        question is a string, valid_answers is a list
    for x in range(n):
       reply = input(Question + '? ')
       if reply == valid_answers.lower():
          return
       else:
          return False
    return reply

option = getReply('Show next option? ', ['yes', 'no', 'y', 'n'])
```

List the errors. You do NOT need to provide a corrected version.

[6 marks]

Question 4c
Learnings: Pay close attention to variable name capitalisation (I only found error 2 through testing)
answer:
1. 'n' is not defined on line 3; the range(n) function will fail.
2. 'Question' is not defined in the function, this will cause it fail. (the function uses 'question' - with a lowercase 'q' as the parameter).
3. reply does not have .lower() on line 5; case-independency will not be applied
4. reply == valid_answers.lower() will fail the equality check on line 5; they are comparing a string to a list rather than a string to a string.
5. .lower() can't be applied to the list valid_answers on line 5; the .lower() function will fail.
6. There is no parameter after return on line 6, the function will return None instead of a valid answer.
7. return reply on line 9 can't be reached because it outside of the if/else block.
8. A question mark is added to the question parameter on line 11; this will add unnecessary duplication of the question mark when input(Question + '? ') is called on line 5.
for testing:

```
def getReply(question, valid_answers):
   #             question is a string, valid_answers is a list
   for x in range(n):
      reply = input(Question + '? ')
      if reply == valid_answers.lower():
         return
      else:
        return False
      return reply

print(option = getReply('Show next option? ', ['yes','no','y','n']))
```

Question 5 **[16 marks]**

(a) **Write a function that builds a dictionary from user input.**
The function repeatedly asks the user for a *key* and an *input-value*, until ***quit*** is entered for the key. The function then returns the dictionary *without* adding *quit* to the dictionary.

The key given by the user may be a number OR a string. if possible, the key should be converted to a floating point number. If this fails, just use string for the key.

All input-values entered by the user are left as strings and are stored in a list associated with the key. This means the user can add several input-values for the same key however only new input-values for a key stored. Duplicates are ignored.

```
key  : color
value: red
key  : 3.5
value: 68
key  : color
value: blue
key  : color
value: red
>> duplicate - ignored
key: quit
```

Would result in the dictionary: `{ 'color':['red', 'blue'], 3.5:['68'] }`

[8 marks]

```
# Question 5a
# Learnings: make sure to put a ':' after declaring a function.
# answer:
def build_dict():
    dict = {}
    while True:
        key = input('key  : ')
        if key == 'quit':
            return dict
        try:
            key = float(key)
        except:
            pass
        value = input('value: ')
        if key not in dict:
            dict[key] = [value]
        else:
            if value in dict[key]:
                print('>> duplicate - ignored')
            else:
                dict[key].append(value)
# for testing:
print(build_dict()) #args color, red, 3.5, 68, color, blue, color, red, quit should display
{ 'color': ['red', 'blue'], 3.5: ['68']}
```

Question 5 continued

(b) Write a program that reads lines from a file *data.txt*, where each line must contain data items (fields) separated by a comma (',') OR a comment line, indicated by a leading **#.**

Your program should count the number of entries and display the total of the credit values. Comment lines are ignored

e.g. For a file containing

```
# Name,   Credit,   Contact
Felicity,373.25,Felicity@hotmail.com
Arthur,xxx,Art@gmail.com
William,211.14,@Bill
```

your program would display something like

```
There were 3 Entries
The total credit is $584.39
```

Important: Lines that don't contain valid 'Credit' values should **not** cause run-time errors. Checking just for 'xxx' isn't sufficient, as there might be other invalid entries. Your program has to handle *all* failed conversions, so the program does not terminate early.

[8 marks]

```
# Question 5c
# Learnings: when doing calculations (in this case, adding credit values), always make sure you have an int() or float() as appropriate.
# answer:
entries = 0
credit = 0
with open('data.txt','r') as f:
    for line in f:
        if line.lstrip()[0] != '#':
            entries += 1
            try:
                credit += float(line.split(',')[1])
            except:
                pass
print('There were {} Entries'.format(entries)) # should be There were 3 Entries
print('The total credit is ${:.2f}'.format(credit)) # should be The total credit is $584.39
```

Question 6 **[11 marks]**

(a) Given a string:

e.g. `s = "It was freezing outside but the cake in the oven would help"`

Write a program to find:

1. the number of words in the string
2. the longest and shortest word
3. the number of words containing the letter 'o'.
4. the total number of characters, excluding spaces.

[8 marks]

```
# Question 6a
# Learnings: it is easy to get = and == mixed up, especially when using them alternatingly.
With every use, think carefully if you are comparing (==), or assigning(=).
# for testing:
s = 'It was freezing outside but the cake in the oven would help'
# answer:
words_in_string = len(s.split()) # should be 12
longest = 0
all_longest = [] # should be 'freezing'
shortest = 1000
all_shortest = [] # should be ['it', 'in]
words_with_o = 0 # should be 3
total_ch_excl_sp = 0 # should be 48
for word in s.split():
   if len(word) > longest:
      longest = len(word)
      all_longest = [word]
   elif len(word) == longest:
      all_longest.append(word)
   if len(word) < shortest:
      shortest = len(word)
      all_shortest = [word]
   elif len(word) == shortest:
      all_shortest.append(word)
   if 'o' in word:
      words_with_o += 1
   total_ch_excl_sp += len(word)
# for testing:
print(words_in_string,all_longest,all_shortest,words_with_o,total_ch_excl_sp)
```

Question 6 continued

(b) When storing data, it's possible to use either lists or dictionaries, however when searching for items, finding items in a list can be much slower that finding them in a dictionary.

Explain why this is so, including explaining how dictionaries achieve an almost constant lookup time. What useful properties do lists have that dictionaries do not?

[3 marks]

Question 6b
answer:
list lookup times can be much slower than dictionary lookup times because lists are searched by index in a linear fashion, meaning indices are evaluated iteratively until a match for the lookup term is found. This means the lookup time is proportional to the length of the list, if the index is not known. This can be expressed as O(n) time complexity.

Conversely, dictionaries achieve an almost constant lookup time because they use the hash of a key in a key value pair to generate the index where the key value pair will be stored in a hash table. Therefore, to retrieve the key value pair from the hash table, the index can be found by running the hash function on the key, and so the key value pair can be found immediately, so long as a good hash function is used so that there are few collisions. Thus, when searching by key, dictionaries have an almost constant look up time, which can also be expressed as O(1) time complexity.

However, lists have a feature that makes them more useful for some tasks: lists are ordered, whereas dictionaries are not ordered. So for instance, it is faster to retrieve the most recently watched movie from a list (because it will be the last item in a list, so the index is known) than from a dictionary. To retrieve it from a dictionary, the dictionary would need the 'most recently watched' status as a tag or a timestamp in the values associated with the movie keys, so every key in the dictionary will need to be evaluated to determine which one was the most recently watched movie - this could be significantly slower than just accessing the last index in a list.

+ + + + + + + +

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 largest item.
max(a, b, c, ...) → object	Return the largest of two or more arguments.
min(iterable) → object	With a single iterable argument, return smallest item.
min(a, b, c, ...) → object	Return the smallest of two or more arguments.
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	
open(filename, mode) → F (a file handle)	open a file, return the file handle
with open(filename, mode) as F:	open a file within a *with* context
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 & 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	
L.remove(value) → NoneType	Remove the first occurrence of value from L
L.reverse() → NoneType	Reverse *IN PLACE*.
L.sort() → NoneType	Sort the list in ascending order.
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.