1601/159.171 ASR
MAN/ALB 2Hr
Distance/Internal NSB

MASSEY UNIVERSITY
MANAWATU AND ALBANY CAMPUSES

EXAMINATION FOR

159.171 COMPUTATIONAL THINKING AND SOFTWARE DEVELOPMENT

Semester One – 2016

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	17	
3	7	
4	18	
5	15	
6	11	
Total	**80**	

Question 1 **[12 marks]**

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

```
print ("12"*3)
print ("Good Morning".split())
print ( len( [1, '+', 2, '+', 3] ) )
```

[3 marks]

```
# Question 1a
# answer
# 121212
# ['Good', 'Morning']
# 5
# for testing:
print("12"*3)
print("Good Morning".split())
print(len([1,'+',2,'+',3]))
```

(b) Translate the following equation into Python, where the $\mathbf{d_1 d_2}$ term indicates that $\mathbf{d_1}$ is multiplied by $\mathbf{d_2}$

$$f = G(\frac{(d_1 d_2)^3}{r^2})$$

[3 marks]

```
# Question 1b
# for testing:
G = 10
d1 = 2
d2 = 4
r = 3
# answer:
f = G*((d1*d2)**3 / r**2)
# for testing:
print(f) # f should be about 500
```

Question 1 continued

(c) If one metre is 39.371 inches, write code to display the height of *Sir Bloggs* in meters, where the title (*Sir*), the name (*Bloggs*), and the height in inches (*67*) are all extracted from the list *data*:

```
data = ["Sir", 'Bloggs', 67]
```

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

```
Sir Bloggs is (67 inches) is 1.7m tall.
```

Write your code so that the height in metres will be displayed rounded to **one decimal place**.

[3 marks]

```
# Question 1c
# for testing:
data = ["Sir", 'Bloggs', 67]
# answer:
print('{} {} is ({} inches) is {:.1f}m tall.'.format(data[0],data[1],data[2],data[2] / 39.371))
# for testing: code above should display 'Sir Bloggs is (67 inches) is 1.7m tall.'
```

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

```
s = "4"
n = 4

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

[3 marks]

```
# Question 1d
# answer:
# 4444
# 12
# 444
# for testing:
s = "4"
n = 4
print(s*int(s))
print(int(s)*3)
print(str(n)*3)
```

Question 2 **[17 marks]**

(a) **Write a Python loop** that prints all the numbers from 15 down to (and including) zero in steps of 5, each on a separate line.

e.g.

```
15
10
5
0
```

[3 marks]

```
# Question 2a
# answer:
for i in range(15,-1,-5):
    print(i)
# for testing: should display the following:
# 15
# 10
# 5
# 0
```

(b) **Using a *while* loop**, write a Python program that, for each element *X* of the list L, displays "*X* contains A" or "*X* does NOT contain A", as appropriate.

so for

```
L = [ "Alphabet", "car"]
```

it displays:

```
Alphabet contains an A
car does not contain an A
```

[5 marks]

```
# Question 2b
# for testing:
L = [ "Alphabet", "car"]
# answer:
count = 0
while count < len(L):
    if 'A' in L[count]:
        print('{} contains an A'.format(L[count]))
    else:
        print('{} does not contain an A'.format(L[count]))
    count += 1
# for testing: # should display the following:
# Alphabet contains an A
# car does not contain an A
```

Question 2 continued

(c) **Write a program** that takes a list L and gives another a list D with the list elements converted to numbers where possible.

e.g if the input list was:

```
L = [ "29.3", "tea", "1", None, 3.14 ]
```

The output list D would contain:

```
D = [ 29.3, "tea", 1, None, 3.14 ]
```

[5 marks]

```
# Question 2c
# Learnings: Type conversion is a surprisingly complex process. It is not sufficient to check the type of the existing argument. Also, 'None' and other types might throw errors. Therefore, a try/except statement is needed. Furthermore, anything that might break the code should go in the try statement. Lastly, think about how the try statement should distinguish between int and float (just check for '.', nothing fancy required).
# for testing:
L = ["29.3", "tea", "1", None, 3.14]
# answer:
D = []
for item in L:
    try:
        if '.' in item:
            D.append(float(item))
        else:
            D.append(int(item))
    except:
        D.append(item)
# for testing:
print(D) # should display: [ 29.3, "tea", 1, None, 3.14]
```

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

```
for i in range(1,3):
    for j in ['hot', 'soup']:
        print(i, j*i)
```

[4 marks]

```
# Question 2d
# answer:
#1 hot
#1 soup
#2 hothot
#2 soupsoup
# for testing:
for i in range(1,3):
    for j in ['hot', 'soup']:
        print(i, j*i)
```

Question 3 **[7 marks]**

(a) There are five variables (with the noted values) that relate to an apartment's desirablity:

1.	distanceToShops	(<500m is Close)
2.	hasInternet	(True/False)
3.	busStopDistance	(< 500m is Close)
4.	libraryWifiAccess	(True/False)
5.	rent	(‘high’, 'medium', ' low')

Write a sequence of statements that will set the variable ***apply*** to *"yes"*, *"maybe"* ,*"no"* or *"don't know"* according whichever of the following rules is satisfied first:

- *"yes"* if the apartment has Internet access, and is close to both the bus and shops
- *"maybe"* if the apartment is not close to the shops but has either Internet access or is close to library Wifi; and rent is low or medium;
- *"no"* if the rent is high, there is no internet access, and it's not close to bus or shops
- if none of the above are satisfied, set *apply* to "don't know"

[5 marks]

```
# Question 3a
# for testing:
distanceToShops = 600
hasInternet = False
busStopDistance = 600
libraryWifiAccess = False
rent = 'high'
# answer:
d = distanceToShops
h = hasInternet
b = busStopDistance
l = libraryWifiAccess
r = rent
if h and b < 500 and d < 500:
    apply = 'yes'
elif d >= 500 and (h or l) and (r == 'low' or r == 'medium'):
    apply = 'maybe'
elif r == 'high' and not h and not (b < 500 or d < 500):
    apply = 'no'
else:
    apply = "don't know"
# for testing:
print(apply) # should display 'no'
```

Question 3 continued

(b) One of the print statements below displays a number but the the other doesn't due to an error in one of the functions. What is the error and how would you fix it?

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

# return the number or for some values, double the number
def testNumber(number):
    result = number
    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:
the print statement - print(first_result) will not display a number, it will display 'None'. This is because the testNumber function has its 'return result' statement as part of the if statement, which only catches numbers between 7 (inclusive) and 20 (exclusive). To fix the error so that it behaves according to the comment, "return the number or for some values, double the number", the 'return result' statement needs to be moved one tab to the left.
for testing:

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

def testNumber(number):
    result = number
    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 **[18 marks]**

(a) **Write a function *getResponse(msg, typeOfResult)***. Both parameters - *msg* and *typeOfResult* - are strings.

If *typeOfResult* is the string *"getString"*, display *msg* and return whatever the user enters as a reply as the return value.

If *typeOfResult* is *"getList"*, display *msg* once, then build a list of the strings from each reply entered by the user. When they enter **`quit`**, return the list of replies collected so far, but ***do not*** include **`quit`** in the returned list.

[6 marks]

```
# Question 4a
# answer:
def getResponse(msg, typeOfResult):
    if typeOfResult == 'getString':
        return input(msg)
    if typeOfResult == 'getList':
        list = []
        print(msg)
        while True:
            response = input('>> ')
            if response == 'quit':
                return list
            else:
                list.append(response)
# for testing:
print(getResponse(msg='fav animal: ', typeOfResult='getString')) # should print 'cat'
print(getResponse(msg='fav animals: ', typeOfResult='getList')) # should print ['cat','dog']
```

Question 4 continued

(b) **Write a function called *checkSpelling (word, 'commonMistakes.txt')*.** The first parameter is a word to check, the second is the name of a file of common mistakes.

Each line of the file has two words separated by a comma. The first word is a common misspelling, the second word is the correct version.

e.g.

```
teh,the
virrus,virus
```

If the parameter *word* is one of the misspellings in the file, the function returns the correct spelling. If the word isn't one of the misspellings, the original word is returned.

If the file doesn't exist, no error occurs and the function always returns the original word. The data in the file should be saved, so that it is only loaded once.

[6 marks]

```
# Question 4b
# Learnings: when useing open(), make sure to give the 'r', or 'w' argument, and also to refer to the file by the variable (such as 'f') after declaring that variable.
# answer:
def checkSpelling(word, commonMistakes=False):
    if commonMistakes:
        with open(commonMistakes,'r') as f:
            for line in f:
                if word == line.strip().split(',')[0]:
                    return line.strip().split(',')[1]
    return word
# for testing:
with open('mistakes.txt','w') as f:
    f.write('teh,the\nvirrus,virus')
print(checkSpelling(word='teh',commonMistakes='mistakes.txt')) # should display 'the'
print(checkSpelling(word='virrus',commonMistakes='mistakes.txt')) # should display 'virus'
print(checkSpelling(word='viras',commonMistakes='mistakes.txt')) # should dispaly 'viras'
print(checkSpelling(word='horse',commonMistakes='mistakes.txt')) # should return 'horse'
print(checkSpelling(word='cat')) # should return 'cat'
```

Question 4 continued

(c) You have two lists, A and B. Without using Python Sets, write code to create three other lists: *Aonly*, *Bonly* and *Both*, that contain items that exist only in list A, only in list B, and in both A and B. The original lists A and B should not be altered.

[6 marks]

```
# Question 4c
# for testing:
A = ['cat','dog','shoe']
B = ['dog','shoe','horse']
# answer:
Aonly = []
Bonly = []
Both = []
for item in A:
    if item in B:
       Both.append(item)
    else:
       Aonly.append(item)
for item in B:
    if item not in A:
        Bonly.append(item)
# for testing:
print(Aonly) # should print ['cat']
print(Bonly) # should print ['horse']
print(Both) # should print ['dog', 'shoe']
```

Question 5 **[15 marks]**

(a) Using *input()*, **write a function that takes a list of words as a parameter,** and lets the user enter a word or a sentence. Any words in the user input that aren't already in the list of words are added and the (possibly longer) wordlist is returned as the result of the function. The test to see if the word is already in the list is NOT case-sensitive.

[5 marks]

```
# Question 5a
# Learnings: be careful when generating a new list; use [:] to shallow copy the list if a new list needs to be returned that doesn't modify the original.
# Always make sure your builtin fuctions have (), as in .lower()
# answer:
def build_sent(listOfWords):
    newList = listOfWords[:]
    response = input('enter word or sentence: ')
    for word in response.split():
        if word.lower() not in [x.lower() for x in newList]:
            newList.append(word)
    return newList
# for testing:
print(build_sent(listOfWords=['cat','dog'])) # response as 'the Cat and Dogs' should display a new list as ['cat','dog','the','and','Dogs']
```

(b) Demonstrate how you would call your function repeatedly until the list contained at least ten words. Then, outside the function, print the list in alphabetical order and exit the program. This is the only time that the list of words is printed.

[5 marks]

```
# Question 5b
# for testing:
def build_sent(listOfWords):
    newList = listOfWords[:]
    response = input('enter word or sentence: ')
    for word in response.split():
        if word.lower not in [x.lower for x in newList]:
            newList.append(word)
    return newList
# answer:
while True:
    newList = build_sent(listOfWords=['cat','dog'])
    if len(newList) >= 10:
        print (sorted(newList))
        break
# for testing: call the above function with varying sentence lengths, it should only break and print if the the new list is equal or greater than 10 words. I've interpreted the question as generating a new list with each function call, rather than adding the list with each function call (which would be recursive, which was not covered in 159.171)
```

Question 5 continued

(c) Write a program that reads lines from a file *numbers.txt*, where a line must contain either:

- two numbers, OR
- a comment line, indicated by a leading #

Your program should add the values of the numbers on each line, ***but only when the first value is larger than the second***. If this is so, the the total of the values on that line are added to an overall running total. Comment lines are ignored.

e.g. for a file containing

```
# Only add lines when first value is larger
1    7
20   1
5   10
40   2
```

Your program would display something like

```
There were 4 lines of numbers
The sum of the 2 selected lines is 63
```

[5 marks]

```
# Question 5c
# for testing:
with open('numbers.txt','w') as f:
    f.write('# Only add lines when first value is larger\n1   7\n20   1\n5   10\n40   2')
# answer
lines = 0
selected_lines = 0
sum = 0
with open('numbers.txt','r') as f:
    for line in f:
        if line.lstrip()[0] != '#':
            lines += 1
            leftD = int(line.split()[0])
            rightD = int(line.split()[1])
            if leftD > rightD:
                selected_lines += 1
                sum += leftD + rightD
print('There were {} lines of numbers'.format(lines))
print('The sum of the {} selected lines is {}'.format(selected_lines,sum))
# for testing should display:
# There were 4 lines of numbers
# The sum of the 2 selected lines is 63
```

Question 6 **[11 marks]**

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

e.g.

```
mp3list = [
   ['Louis Armstrong', 'Ella and Louis Again', "Don't Be That Way"],
   ['Louis Armstrong', 'Ella and Louis Again', 'They All Laughed' ],
   ['Louis Armstrong', 'Porgy and Bess',       'Summertime'       ],
   ['Louis Armstrong', 'Porgy and Bess',    'I Wants to Stay Here'],
   ['Adele',           '25',                   'Hello'],
   ['Adele',           '25',                   'Send my love'],
   ['Adele',           '25',                   'I miss you']]
```

Write a function that takes a string as a parameter and searches for those tracks that contain that string. It then display the details (Artist/Album/ and Track name) of the matching tracks, cleanly formatted, i.e. not just printing the sublist(s).

[5 marks]

```
# Question 6a
# Learnings: don't over-think how to look through the index of a nested list. Just look through the index of each sublist as you iterate over the sublists.
# answer:
def find_tracks(track_string):
    for entry in mp3list:
        if track_string in entry[2]:
            print('Artist: {:>10}, Album: {:>10}, Track: {>10}'.format(entry[0],entry[1],entry[2]))
# for testing:
mp3List = [['Louis Armstrong', 'Ella and Louis again', "don't be that Way"],['Louis Armstrong', 'Porgy and Bess', 'Summertime']]
find_tracks(track_string='Summer') # should display one result
find_tracks(track_string='t') # should display two results
```

Question 6 continued

(b) When saving data, it's possible to use either lists or dictionaries, however when searching for items, finding items in a list is usually much slower that finding them in a dictionary. **Using examples, 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?

[6 marks]

Question 6b
answer:
Lists usually have much slower lookup times than dictionaries because the lookup is a linear search; that means that each index is checked for a match until a match is found. Therefore list lookup times are proportional to the size of the list (this can also be expressed as O(n) time complexity. Conversely, dictionaries use a hash function based on the key to store the key value pair in a hash table, so a key value pair can be directly retrieved from its index in the hash table by running the hash function on the key (so long as it is a good hash function, and few collisions occur). This makes dictionaries typically much faster to lookup than than lists, because the index is usually already known based on the key, so the lookup times are not necessarily proportional to the size of the dictionary (this can also be expressed as O(1) time complexity).

For example, lists or dictionaries could be used for storing a table of movies watched. If you wanted to know if the movie 'Kill Bill' had been watched, then in a list of movies (['Gran Turino', 'The Matrix', 'Kill Bill']) each index would need to be checked for a match. However, in a dictionary ({'Gran Turino': 'watched', 'The Matrix': 'watched', 'Kill Bill': 'watched'}), the movie 'Kill Bill' could be found immediately based on the hash of the key 'Kill Bill'. However, lists are ordered, whereas dictionaries are not ordered, which makes lists more useful for operations where order is desirable. For instance, to find the most recent movie watched, the movie can be directly accessed as the last index in the list. However, to know what the last movie watched was in a dictionary, the dictionary would need to add additional information about the time each movie was watched, and every index in the dictionary would need to be searched to determine which movie had the most recent viewing time stamp.

+ + + + + + + +

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	Insert x at position indexed by index
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.