THE UNIVERSITY OF AUCKLAND

Summer Semester, 2018
Campus: City

COMPUTER SCIENCE
SOLUTIONS
Principles of Programming

(Time Allowed: TWO hours)

NOTE:
You must answer **all** questions in this exam.
No calculators are permitted.
Answer in the space provided in this booklet.
There is space at the back for answers which overflow the allotted space.

Surname	
Forenames	
Preferred Name (if different to forenames)	
Student ID	
Username	

Q1 (/10)	**Q4** (/15)	**Q7** (/15)
Q2 (/14)	**Q5** (/9)	**Q8** (/10)
Q3 (/12)	**Q6** (/15)	**TOTAL** (/100)

Question 1 (10 marks)

a) Complete the output produced by the following code.

```
a = 2 * 3 ** 2 + 1 - 9 // 2 * 2
b = 4 % 10 * 10 % 5
print("a:", a, "b:", b)
```

```
a:    11      b: 0
```

(2 marks)

b) Complete the output produced by the following code.

```
words = "CHIT-CHAT"
index1 = words.find('T')
index2 = words.rfind('H')
letters = words[index1: index2]
print("Letters:", letters)
```

```
Letters: T-C
```

(2 marks)

c) In the boxes below, assign integer values to the variable, `value`, so that the output when the code is executed is:

```
Yes
No
```

`value =` **5**

#a single digit number which is either exactly divisible by 5 or exactly divisible by 4.

```
if len(str(value)) == 1 and (value % 5 == 0 or value % 4 == 0):
    print("Yes")
else:
    print("No")
```

`value =` **21**

#a number which has more than one digit or is not exactly divisible by 5 and is not exactly divisible by 4.

```
if len(str(value)) == 1 and (value % 5 == 0 or value % 4 == 0):
    print("Yes")
else:
    print("No")
```

(2 marks)

d) Complete the first assignment statement in the code below so that the output of the code is:

```
Result: True
```

```
a_string = "A B D X B X"          #a string which has exactly six tokens and the second to last token must be a 'B'.

a_list = a_string.split()
result = len(a_list) == 6 and a_list[1] == a_list[-2]
print("Result:", result)
```

(2 marks)

e) Complete the output produced by the following code.

```
list1 = [1, 2, 5]
list2 = list1
list1.append(6)

tuple1 = (5, 7)
tuple2 = tuple1
tuple1 = tuple1 + (4, 6, 1)

print("1:", list1 == list2, "2:", tuple1 == tuple2)
```

```
1:  True  2: False
```

(2 marks)

Question 2 (14 marks)

a) Complete the output produced when the following `main()` function is executed.

```
def main():
    total = do_something([3, 4, 2, 6, 7, 1], 4)
    print("1:", total)
    print("2:", do_something([3, 4, 2, 6, 7, 1], 2))

def do_something(a_list, position):
    a_list.sort()
    total = 0
    for i in range(position, len(a_list)):
        total = total + a_list[i]

    return total
```

```
1: 13
2: 20
```

(4 marks)

b) In the `main()` function below, complete the function call so that the output when the `main()` function is executed is:

```
Result: 6
```

```
def main():
    a_list = [4, 3, 2, 6, 0, 1]

    result = do_something(a_list, 5)

    print("Result:", result)

def do_something(a_list, position):
    a_list.sort()
    total = 0
    for i in range(position, len(a_list)):
        total = total + a_list[i]

    return total
```

(4 marks)

c) Complete the `get_unique_code()` function below which returns a string which is made up of four **unique** random digits from 2 to 8 both inclusive. Note that the string which is returned must have a length of 4 and all four digits in the string must be unique.

Note: you can assume that the `random` module has been imported.

For example, executing the following `main()` function using the completed `get_unique_code()` function may give the output:

```
1. 7635
2. 3758
3. 6843
```

```
def main():
    code = get_unique_code()
    print("1.", code)
    print("2.", get_unique_code())
    print("3.", get_unique_code())
```

```
def get_unique_code():
```

```
allowed_digits = "2345678"
code = ""

while len(code) < 4:
    index = random.randrange(
                        len(allowed_digits))
    digit = allowed_digits[index]

    if digit not in code:
        code = code + digit

return code
```

(6 marks)

Question 3 (12 marks)

a) Give the output produced when the following `main()` function is executed.

```
def main():
    function_ifs(4, 6, 3)

def function_ifs(a, b, c):
   if a > b and c < b:
      print("A")
   elif not a < 5 and a > c:
      print("B")
   elif b < c or c < 5:
      print("C")
      if not a < 20:
          print("D")
      else:
          print("E")
          if a > b or c > b:
              print("F")
   if a >= 4:
      print("G")
   print("H")
```

C
E
G
H

(4 marks)

b) Give the output produced when the following code is executed.

```
number = 10

while number > 3:
   if number % 3 == 0:
      number = number - 1
   print(number, end = " ")
   number = number - 1

print(number)
```

```
10 8 7 5 4 3
```

(4 marks)

c) Rewrite the following code using an equivalent `for ... in range()` loop instead of the `while` loop:

```
num = 34
count = 0
while num >= 10:
   print(num)
   count = count + 1
   num = num - 5
```

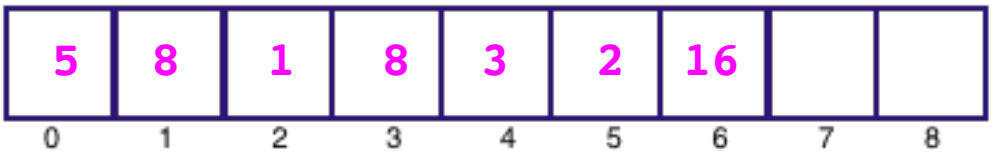
```
count = 0

for num in range(34, 9, -5):
    print(num)
    count = count + 1
```

(4 marks)

Question 4 (15 marks)

a) In the boxes below, show each element of `a_list` after the following code has been executed. Use as many of the boxes as you need.

```
a_list = [5, 3, 1, 8, 2, 4]

a_list.insert(2, 8)
value = a_list.pop(1)
a_list.insert(4, value)
value = a_list.pop()
value = value * a_list.index(3)
a_list.append(value)
```



(4 marks)

b) Complete the output produced when the following `main()` function is executed.

```
def main():
    a_list = [4, 3, 2, 9, 2, 7, 8, 3, 2, 8]

    count = 0
    for index in range(len(a_list)):
        num = a_list[index]
        if num in a_list[index + 1:]:
            count = count + 1

    print("Count:", count)
```

```
Count: 4
```

(5 marks)

c) Complete the `get_funny_sum()` function which is passed two list parameters of exactly the same length. The first parameter is a list of integers and the second parameter is a list of booleans. The function returns a "total" which is obtained in the following way:

> Each value of the list of integers is added to the total if the corresponding boolean is `True` and it is subtracted from the total if the corresponding `boolean` is `False`.

For example, executing the following `main()` function using the completed function gives the output:

```
Total: 9
```

```
def main():
    a_list = [4, 2, 1, 5, 3]
    add_or_subtract = [True, False, False, True, True]
    total = get_funny_sum(a_list, add_or_subtract)
    print("Total:", total)

def get_funny_sum(numbers, plus_minus):
```

```
total = 0

for i in range(len(numbers)):
    if plus_minus[i]:
        total = total + numbers[i]
    else:
        total = total - numbers[i]

return total
```

(6 marks)

Question 5 (9 marks)

a) Complete the output produced when the following `main()` function is executed.

```
def main():
    a_list = [3, 4, 7]
    do_something1(a_list)
    print("a_list:", a_list)

def do_something1(list1):
    list2 = list1
    extra_list = [4, 5]
    for element in extra_list:
        list2.append(element)
```

```
a_list: [3, 4, 7, 4, 5]
```

(2 marks)

b) Give the output produced when the following `main()` function is executed.

```
def main():
    a_list = [3, 7]
    do_something2(a_list)
    print("a_list:", a_list)

def do_something2(list1):
    list2 = [4, 6, 2]
    for element in list1:
        list2.insert(2, element)
    print("list2:", list2)
    list1 = list2
```

```
list2: [4, 6, 7, 3, 2]
a_list: [3, 7]
```

(2 marks)

c) Given the following code, what is the type of each of the three Python objects (`object1`, `object2` and `object3`)?

```
a_list = [1, 6, "four"]
a_tuple = (3, "funny", True)
a_dict= {"funny": (3, 5), "odd": (2, 1)}

object1 = a_list.pop(1)
object2 = a_tuple[a_list[0]]
object3 = a_dict[a_tuple[1]]
```

```
object1 is of type: int
object2 is of type: string
object3 is of type: tuple
```

(3 marks)

d) The following function contains a docstring. In the docstring, add **one** doctest which does not fail.

```
def do_a_check(value1, value2, value3):
    """Checks the parameter values
```

```
>>> do_a_check([4, 6, 7], 1, 6)
True
>>> do_a_check("6145", 3, 5)
True
```

(2 marks)

```
    """
    number = int(value1[value2])
    return number == value3

import doctest
doctest.testmod()
```

Question 6 (15 marks)

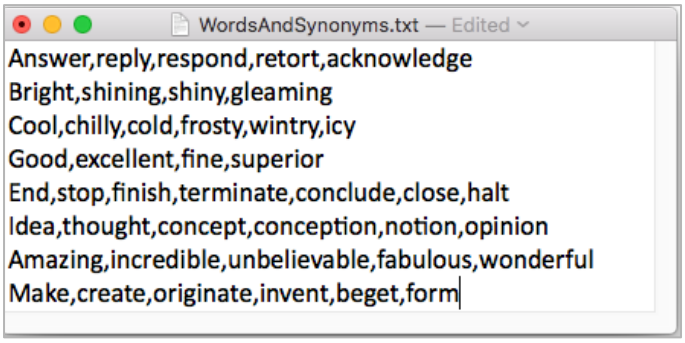
Initial Note: a synonym is a word which has the same or nearly the same meaning as another word.

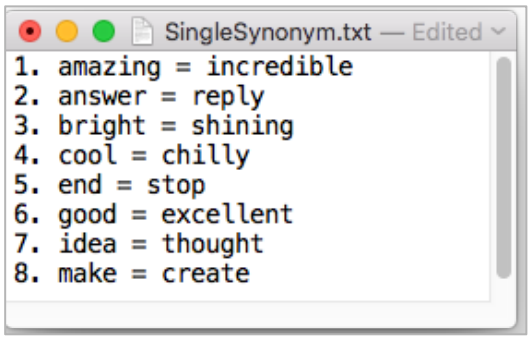
The following program reads information from the `"WordsAndSynonyms.txt"` file, processes the information and writes a simpler version of the information to the `"SingleSynonym.txt"` file.

Each line of the input file is made up of a word followed by one or more synonyms of the word separated by commas, e.g.,

```
"Amazing,incredible,unbelievable,fabulous,wonderful"
```

Below is an example of a `"WordsAndSynonyms.txt"` file (on the left) and the corresponding `"SingleSynonym.txt"` file (on the right) produced by the completed program:





a) Complete the `get_word_synonyms_list()` function which is passed one parameter: the name of a file containing words and their synonyms. Each line of the input file is one word followed by one or more synonyms separated by commas. The function returns a **sorted** list of strings. Each element of the returned list corresponds to each line of the input file all in lowercase characters and should not contain any newline characters.

b) Complete the `remove_extra_synonyms()` function which is passed a list of strings as a parameter. Each element of the parameter list is made up of a word followed by one or more synonyms of the word, separated by commas. This function changes the string elements of the parameter list so that each element is a **tuple** made up of the first word followed by the first synonym, e.g., the element:

`"amazing,incredible,unbelievable,fabulous,wonderful"` becomes the tuple `("amazing", "incredible")`.

The first two lines of this function have been completed for you.

c) Complete the `write_to_file()` function which has two parameters: the name of the output file and a list of tuples where each tuple contains two string elements: one word followed by one synonym. This function writes a numbered list of the word from each parameter list tuple followed by `" = "` followed by the synonym. Note that each number is followed by the string `". "`. See the screenshot of the example output file above on the right.

```
def main():
    filename = "WordsAndSynonyms.txt"
    words_and_synonyms = get_word_synonyms_list(filename)
    remove_extra_synonyms(words_and_synonyms)
    write_to_file("SingleSynonym.txt", words_and_synonyms)

def get_word_synonyms_list(filename):
    file_in = open(filename, "r")
    contents = file_in.read()
    file_in.close()
    contents = contents.lower()
    word_synonyms_list = contents.split("\n")
    word_synonyms_list.sort()
    return word_synonyms_list

def remove_extra_synonyms(words_and_synonyms):
    for i in range(len(words_and_synonyms)):
        single_string = words_and_synonyms[i]
        line_list = single_string.split(",")
        word = line_list[0]
        one_synonym = line_list[1]
        words_and_synonyms[i] = (word,
                                 one_synonym)

def write_to_file(filename, words_and_synonyms):
    number = 1

    file_out = open(filename, "w")
    for a_tuple in words_and_synonyms:
        line_of_info = str(number) + ". " +
                a_tuple[0] + " = " + a_tuple[1]
                                    + "\n"
        file_out.write(line_of_info)
        number += 1

    file_out.close()

main()
```

(15 marks)

Question 7 (15 marks)

a) Complete the following `main()` function which changes the `a_dict` dictionary in the following way:

Any corresponding values which are greater than their key are changed to the same number as the key,

e.g., if the key: value pair is 6: 8 then this pair becomes 6: 6, if the key: value pair is 6: 3 then this pair is left unchanged. The output of the completed code is:

```
Dictionary: {9: 9, 3: 3, 4: 4, 5: 5, 7: 2}
```

```
def main():
    a_dict = {9: 12, 3: 7, 4: 4, 5: 8, 7: 2}
```

```
for num_key in a_dict:
    value = a_dict[num_key]
    if value > num_key:
        a_dict[num_key] = num_key
```

(4 marks)

```
    print("Dictionary:", a_dict)
```

b) Complete the output produced when the following `main()` function is executed:

```
def main():
    a_dict1 = {'A': 4, 'B': 6, 'V': 2, 'N': 3}
    a_dict2 = {1: 'T', 4: 'M', 6: 'Q', 7: 'R', 3: 'K'}
    dict1_letters = list(a_dict1.keys())
    dict1_letters.sort()
    word = ""

    for letter in dict1_letters:
        value = a_dict1[letter]
        if value in a_dict2:
            word = word + a_dict2[value]
        else:
            word = word + "X"

    print("Word:", word)
```

```
Word: MQKX
```

(4 marks)

c) Complete the `get_dictionary()` function which is passed a list of strings as a parameter. Each string in the parameter list is made up of exactly three letters followed by one or more digits. The function returns a dictionary where:

- the keys are the **unique** strings obtained from the first three letters of the parameter list elements - these are the letter codes,
- the values corresponding to each letter code are **a sorted list** of all the integers which have the same first three letters. The integers are obtained by slicing the list element string from index 3 to the end of the string and then converting it into an integer.

For example, executing the following program with the completed function prints:

```
{'stx': [766], 'pmd': [65, 651], 'abc': [213, 456, 8765]}
```

```
def main():
    codes_list = ["abc8765", "pmd65", "abc456", "abc213",
                                              "pmd651", "stx766"]
    a_dict = get_dictionary(codes_list)
    print(a_dict)

def get_dictionary(codes_list):
    a_dict = {}

    for code_string in codes_list:
        code_key = code_string[:3]
        number = int(code_string[3:])

        if code_key in a_dict:
            if number not in a_dict[code_key]:
                a_dict[code_key].append(
                                              number)
                a_dict[code_key].sort()
        else:
            a_dict[code_key] = [number]

    return a_dict
```

(7 marks)

```
main()
```

Question 8 (10 marks)

Both parts a) and b) of this question refer to the following program:

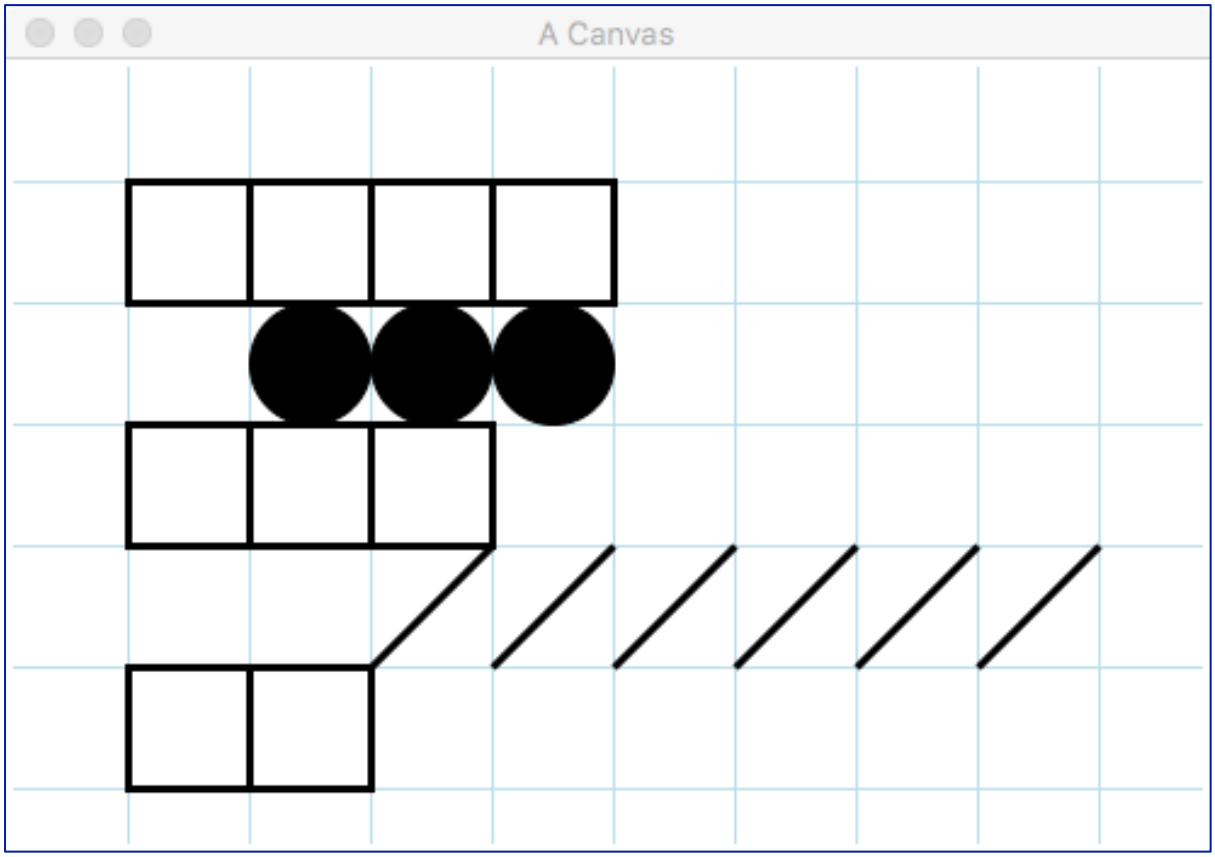
```
from tkinter import *
def draw_pattern(a_canvas):
    size = 10
    pattern_list = [(1, 4), (2, 3), (1, 3), (3, 6), (1, 2)]
    top = size
    for each_tuple in pattern_list:
        shape_type = each_tuple[0]
        how_many = each_tuple[1]
        left = size * shape_type
        for count in range(how_many):
            area = (left, top, left + size, top + size)
            if shape_type == 1:
                a_canvas.create_rectangle(area)
            elif shape_type == 2:
                a_canvas.create_oval(area, fill='black')
            elif shape_type == 3:
                a_canvas.create_line(left+size,top,left,top+size)
            left =  left + size
        top = top + size
def main():
    root = Tk()
    root.title("A Canvas")
    root.geometry("125x85+10+10")
    a_canvas = Canvas(root, bg="white")
    a_canvas.pack(fill=BOTH, expand=1) #Canvas fills whole window
    draw_pattern(a_canvas)
    root.mainloop()

main()
```

a) In the above program, what kind of shape corresponds to a `shape_type` of 3?

A line

(2 marks)

b) As accurately as possible, in the window below, show what is drawn by the above program. Grid lines have been drawn in the window to help you. The gap between adjacent gridlines is 10 pixels.



(8 marks)