THE UNIVERSITY OF AUCKLAND

SEMESTER TWO, 2018
Campus: City

COMPUTER SCIENCE

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

Question 1 (12 marks)

a) Complete the output produced by the following code.

```
a = 5 // 3 + 6 % 8
b = 5.7 // 3 + 16 % 4
print("a:", a, "b:", b)
```

```
a:              b:
```

(2 marks)

b) Complete the output produced by the following code.

```
result = 21 // 4 ** 2 - 1 + 16 / 5 * 3 // 2 - 1
print("Result:", result)
```

```
Result:
```

(2 marks)

c) Complete the output produced by the following code.

```
phrase = "How lovely."
bits_of_string = (phrase[-2] + "-" + phrase[2:5] +
                  "-" + phrase[4: : 2])
print("Output:", bits_of_string)
```

```
Output:
```

(2 marks)

d) Assume the variable, `value`, has been initialised to an integer value. Write a boolean expression which tests if `value` is an odd number which is exactly divisible by 9.

(2 marks)

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

```
def main():
    do_ifs(52, 50)

def do_ifs(num1, num2):
    if num1 < num2 and num2 < 60:
        print("A", end = " ")
        if num2 % 2 == 0:
            print("B", end = " ")
        elif num2 % 3 == 0:
            print("C", end = " ")
        print("D", end = " ")
    else:
        if num1 > 50 or num2 < 50:
            print("E", end = " ")
        if num2 % 2 == 1:
            print("F", end = " ")
        print("G", end = " ")

    print("H")
```

(2 marks)

f) In the docstring of the `get_result()` function below, add a short description (fifteen words or less) of the function.

```
def get_result(number, list_of_numbers):
    """
    """
```

(2 marks)

```
    result_number = list_of_numbers[0]
    smallest_difference = abs(result_number - number)
    for value in list_of_numbers:
        diff = abs(value - number)
        if diff < smallest_difference:
            smallest_difference = diff
            result_number = value

    return result_number
```

Question 2 (12 marks)

a) Using a `while` loop, complete the code below so that the phrase `'Do it again!'` is printed 100 times.

```
while
        print('Do it again!')
```

(2 marks)

b) Complete the `get_user_number()` function below which continuously prompts the user for a positive odd number less than 10 until the number entered by the user satisfies this requirement. You can assume that the user will only enter a whole number. As soon as the user enters a positive odd number less than 10, the function returns the number. Below are two example outputs produced using the completed program. In the examples the user input is shown in a bigger font and in bold.

Enter a positive odd number less than 10: **-3**
Enter a positive odd number less than 10: **25**
Enter a positive odd number less than 10: **9**
User number: 9

Enter a positive odd number less than 10: **7**
User number: 7

```
def main():
    number = get_user_number()
    print("User number:", number)

def get_user_number():
    prompt = "Enter a positive odd number less than 10: "
```

(4 marks)

```
main()
```

c) Using a `for … in` loop, complete the code below so that the phrase `'Do it again!'` is printed 100 times.

```
for

        print('Do it again!')
```

(2 marks)

d) Complete the output produced by the following code.

```
count_a = 0
count_b = 0
count_c = 0
for num1 in range(6):
    count_a += 1
    for num2 in range(3):
        count_b += 1
    count_c += 1
print("A:", count_a, "B:", count_b, "C:", count_c)
```

```
A:        B:        C:
```

(2 marks)

e) Complete the output produced by the following code.

```
count_a = 0
count_b = 0
count_c = 0
for num1 in range(2, 5):
    count_a += 1
    for num2 in range(num1):
        count_b += 1
count_c += 1
print("A:", count_a, "B:", count_b, "C:", count_c)
```

```
A:        B:        C:
```

(2 marks)

Question 3 (12 marks)

a) Complete the `capitalise_letter()` function which is passed two parameters: a string (`word`) and an integer (`index`). The function returns the word all in lowercase characters except for the letter at the index given by the `index` parameter. For example, executing the following program with the completed function prints:

```
1. bloSsom
2. cArrot
3. Dog
```

Note: you may assume that the `index` parameter value is always less than the length of the `word` parameter.

```
def main():
   result = capitalise_letter("BLOSSOM", 3)
   print("1.", result)
   print("2.", capitalise_letter("carrOT", 1))
   print("3.", capitalise_letter("dog", 0))

def capitalise_letter(word, index):
```

(6 marks)

```
main()
```

b) The following program firstly prompts the user for a number between 5 and 20 using the `prompt` variable defined in the `main()` function, then the program calculates the cost of the number of items entered by the user, and finally the program displays the details of the transaction (number of items, cost per item, handling cost and the final price rounded to the closest whole dollar).

All the functions for this program have already been defined. You are required to complete the `main()` function of the program by adding **THREE** lines of code, with each line making a call to one of the three defined functions. Below are two example outputs produced using the completed program (the user input in the examples is shown in a larger bold font).

```
Enter number (5 – 20): 6

Items: 6 Cost per item: $4.25
Handling cost: $5
Total $30
```

```
Enter number (5 – 20): 10

Items: 10 Cost per item: $4.25
Handling cost: $5
Total $48
```

```
def main():
    handling_cost = 5
    cost_per_item = 4.25
    prompt = "Enter number (5 – 20)"
```

```
def get_number(prompt):
    return int(input(prompt + ": "))

def get_cost(number, cost_per_unit, handling_cost):
    total_cost = number * cost_per_unit + handling_cost
    return round(total_cost)

def display_details(items,cost_each,handling_cost,final_price):
    print()
    print("Items: ",items," Cost per item: $",cost_each,sep="")
    print("Handling cost: $", handling_cost, sep="")
    print("Total $", final_price, sep="")

main()
```

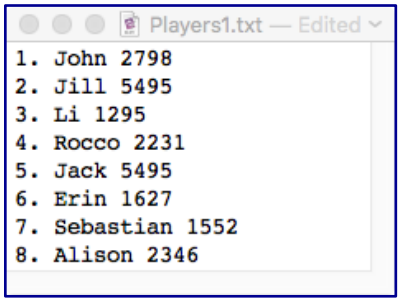
(6 marks)

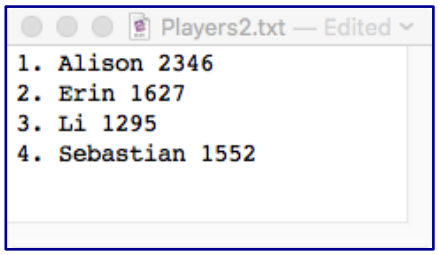
Question 4 (15 marks)

The following program reads a numbered list of players and id numbers from the input file, `"Players1.txt"`, removes some of the players from the list and writes a **sorted** numbered list of players and id numbers to the output file, `"Players2.txt"`.

- The input file contains a numbered list of players' names, each followed by an id number, each player on a separate line.
- The output file displays a **sorted** numbered list of names and id numbers.

Below is an example of a `"Players1.txt"` file (on the left) and the corresponding `"Players2.txt"` file (on the right) produced by the completed program:





a) Complete the `get_players_list()` function which is passed one parameter, the name of a file which contains a numbered list of players and id numbers, one player's information per line. This function returns a list of each line of the input file (a list of strings). Each element of the returned list should not contain a newline character.

b) Complete the `get_updated_players_list()` function which is passed two lists of strings as parameters:

- a list of strings where each element of the list is made up of a number followed by a dot and a space, followed by a player name, a space, and finally, an id number, e.g., `"5. Jack 5495"`.
- a list of id numbers (strings).

The function returns a **new** list of strings containing all the elements from the first parameter list except for those elements which have an id number listed in the second parameter list.

c) Complete the `remove_numbering()` function which is passed a list of strings as a parameter: each string in the parameter list starts with a number, followed by a dot and a space, e.g., `"5. Jack 5495"`.. This function removes the number, the dot and the space from the beginning of each string element of the parameter list.

d) Complete the `write_to_file()` function which has two parameters: the name of a file and a list of strings. Each element of the list is made up of a player name, followed by a space and an id number, e.g., `"Jack 5495"`.. This function first sorts the list and then writes a **numbered** list of the players followed by a space and their id number to the file, each player on a new line (see the example output file above on the right):

```
def main():
    ids_to_remove = ["5495", "2231", "5495", "2798"]
    players = get_players_list("01_Players1.txt")
    players = get_updated_players_list(players, ids_to_remove)
    remove_numbering(players)
    write_to_file("01_Players2.txt", players)
```

```
def get_players_list(filename):
```

```
def get_updated_players_list(players_list, ids_to_remove):
```

```
def remove_numbering(players_list):
```

```
def write_to_file(filename, players_list):
```

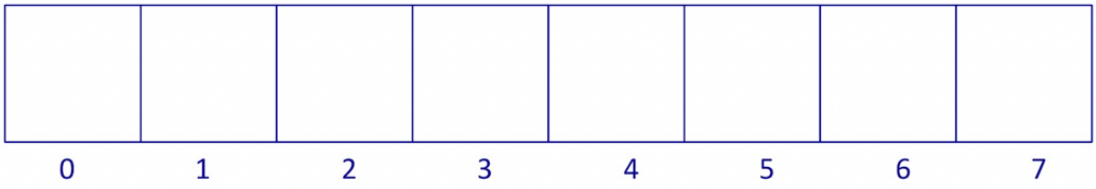
```
main()
```

(15 marks)

Question 5 (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, 7, 5, 14]
a_list = a_list + [11]
a_list.insert(2, 3)
value = a_list.pop(0)
a_list.append(value)
a_list.append(len(a_list))
```



(4 marks)

b) Complete the output produced by the following code.

```
numbers = [5, 7, 8, 3, 2]
value1 = max(numbers)
value2 = min(numbers)

result1 = []
result2 = []
for num in numbers:
    result1.append(value1 - num)
    result2.append(num - value2)

print("Result1:", result1)
print()
print("Result2:", result2)
```

```
Result1: [                    ]

Result2: [                    ]
```

(4 marks)

c) Complete the `get_special_average()` function which is passed a list of integers as a parameter. The function calculates and returns the average of the parameter list of integers, excluding the largest and smallest values in the list. The average which is returned by the function is rounded to one decimal place. For example, executing the following `main()` function using the completed `get_special_average()` function gives the output:

```
Result: 4.5
```

Note: you may assume that the parameter list always contains at least three elements.

```
def main():
    a_list = [5, 7, -8, 5, 14, 1]
    print("Result:", get_special_average(a_list))


def get_special_average(nums_list):
```

(7 marks)

```
main()
```

Question 6 (15 marks)

a) Complete the following code which sorts all the lists corresponding to the keys in the `a_dict` dictionary. The output of the completed code is:

```
AFTER: {'BC618': [19, 45, 65, 98], 'BC235': [38, 39, 56],
        'BC345': [25, 34, 62, 87]}
```

```
a_dict = {'BC345': [34, 25, 87, 62],
          'BC618': [98, 65, 19, 45],
          'BC235': [56, 39, 38]  }









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

(4 marks)

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

```
def main():
    total = 0
    index = 0
    a_dict = {"S": 5, "C": 2, "D": 3, "A": 2, "E": 8}
    codes = "984351AXBCDST"

    for key in a_dict:
        if key in codes and str(a_dict[key]) in codes:
            print(key, "-", a_dict[key])
            total = total + int(codes[index])
            index = index + 1

    print("Total:", total)
```

(4 marks)

c) Complete the `add_dict2_values()` function which is passed two `dict` objects as parameters, `dict1` and `dict2`. Both parameter dictionaries have a single character as the keys and a list of integers as the corresponding values. The function looks at the lists corresponding to the same key in both dictionaries. For any key which is the same in both dictionaries, then any integer in the corresponding list of `dict2` which is not already in the corresponding list of `dict1` is added to the `dict1` corresponding list. All the corresponding lists of `dict1` are kept in **sorted** order.

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

```
A: [1, 2, 3, 5]
B: [1, 2, 4, 7, 8]
X: [0, 9]
N: [3, 8]
```

```
def main():
    dict1 = {"A":[1,2,3,5], "B":[1,2,8], "X":[0,9], "N":[8]}
    dict2 = {"A":[5], "B": [2,4,7], "T": [5,6], "N":[3,8]}
    add_dict2_values(dict1, dict2)
    for key in dict1:
        print(key, ": ", dict1[key], sep = "")

def add_dict2_values(dict1, dict2):
```

(7 marks)

```
main()
```

Question 7 (10 marks)

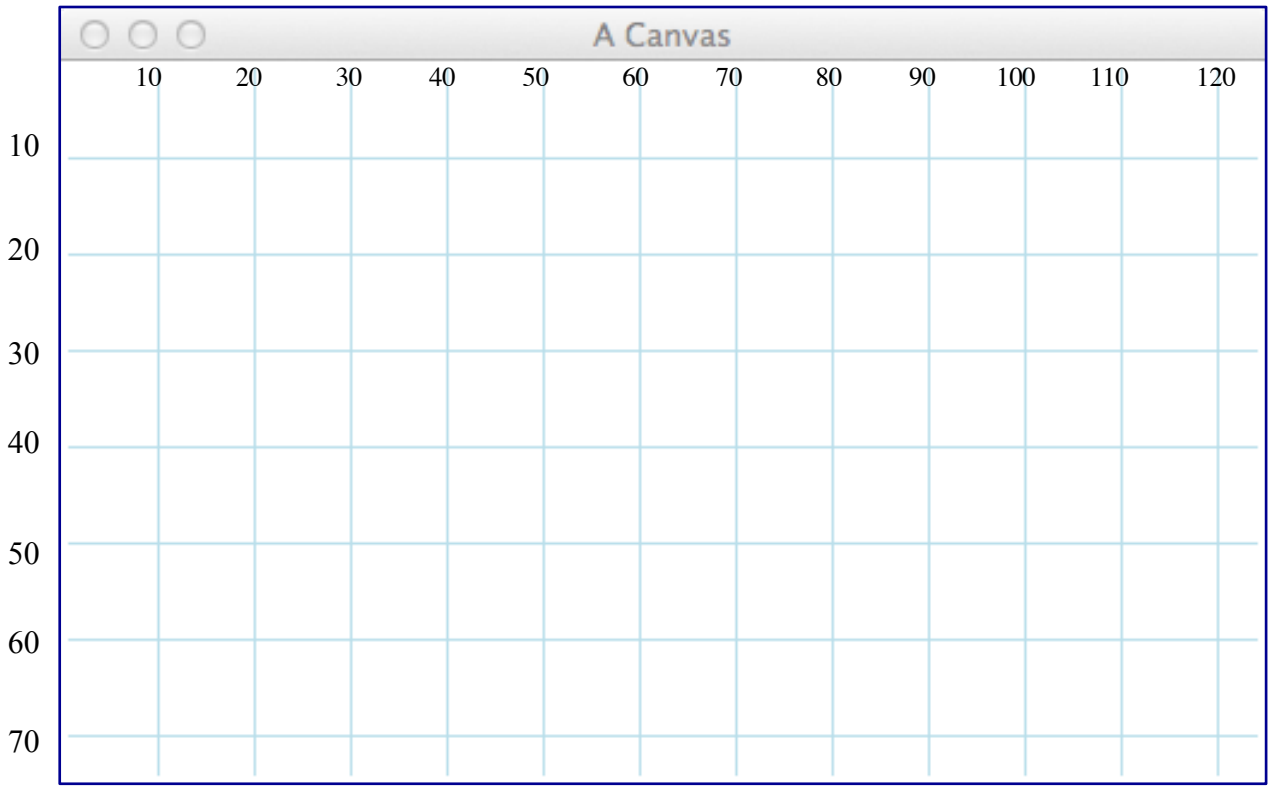
Both parts of this question refer to the following program:

```
from tkinter import *

def draw_pattern(a_canvas, size):
    top = size
    indent_list = [size * 3, size * 2, size]
    pattern_dict = {0: [2, 4, 3, 1],
                    1: [3, 2, 1, 4],
                    2: [1, 1, 3, 4]
                }
    sorted_keys = list(pattern_dict.keys())
    sorted_keys.sort()
    for key in sorted_keys:
        list_of_shapes = pattern_dict[key]
        left = indent_list[key]
        for shape in list_of_shapes:
            if shape == 1:
                a_canvas.create_rectangle(left, top,
                                left + size, top + size)
            elif shape == 2:
                a_canvas.create_line(left, top + size,
                                left + size , top + size)
            elif shape == 3:
                a_canvas.create_oval(left, top, left + size,
                                top + size, fill='black')
            else:
                a_canvas.create_line(left , top, left + size,
                                top + size)
            left = left + size
        top = top + size * 2

def main():
   root = Tk()
   root.title("A Canvas")
   root.geometry("130x75+10+10")
   a_canvas = Canvas(root, bg="white")
   a_canvas.pack(fill=BOTH, expand=1) #Canvas fills whole window
   draw_pattern(a_canvas, 10)
   root.mainloop()

main()
```

a) In the above program, how many rows of shapes are drawn?

(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)

Question 8 (9 marks)

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

```
def main():
    a_list = [4, 7]
    fiddle1(a_list)
    print("a_list:", a_list)

def fiddle1(list1):
   extra_elements = [4, 6, 7, 1]
   list2 = list1
   for element in extra_elements:
      if element not in list1:
          list2.append(element)
```

```
a_list:
```

(2 marks)

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

```
def main():
    a_list = [3, 7, 6]
    fiddle2(a_list)
    print("a_list:", a_list)

def fiddle2(list1):
    list2 = [1, 3]
    list1 = list2
    for i in range(len(list1) - 1, -1, -1):
        list2.append(list1[i])
```

```
a_list:
```

(2 marks)

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

```
a_string = "Hopscotch"
a_dict = {2: "A", 5: "CC", 9: "X"}
a_list = [4, a_dict[2], "st"]

object1 = (a_list[2], a_string[-3])
object2 = a_string.find(a_list[2])
object3 = a_list[1] * 3
```

```
object1 is of type:
object2 is of type:
object3 is of type:
```

(3 marks)

d) In the docstring of the `do_a_check()` function below, add ONE doctest which does not fail.

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

(2 marks)

```
    """
    number1 = value1 + value2
    number2 = value1 * value2

    return number1 < number2

import doctest
doctest.testmod()
```

OVERFLOW PAGE

(If you have used this page, please indicate clearly under the relevant question that you have overflowed to this page)