THE UNIVERSITY OF AUCKLAND

SECOND SEMESTER, 2016
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	**SOLUTIONS**
Forenames	
Student ID	
Username	

Q1 (/15)	**Q4** (/15)	**Q7** (/10)
Q2 (/15)	**Q5** (/15)	**TOTAL** (/100)
Q3 (/15)	**Q6** (/15)	

Question 1 (15 marks)

a) Complete the output produced by the following code.

```
result = 2 + 3 ** 2 * 2 - 3 // 2
print("Result:", result)
```

```
Result:  19
```

(3 marks)

b) Complete the output produced by the following code.

```
place = "HOKITIKA"
place = place[4:] + place[:2] + place[-3]
print("place:", place)
```

```
place:  TIKAHOI
```

(3 marks)

c) Complete the output produced by the following code.

```
def get_convert_num(number, conversion_str):
   result = ""
   num_str = str(number)
   for digit in num_str:
       index = int(digit)
       result = result + conversion_str[index]

   return result

def main():
   print("1.", get_convert_num(101, "stuvwxyzab"))
   print("2.", get_convert_num(251, "abcdefghij"))

main()
```

```
1.  tst

2.  cfb
```

(4 marks)

d) Complete the `add_slice()` function which is passed three parameters:

- `letters` - a string of characters.
- `how_many` - an integer which is the length of the slice which is to be concatenated with the `letters` string.
- `from_front` - a boolean which indicates whether the slice to be added is to be taken from the front or from the end of the `letters` string.

The function takes a slice of the `letters` string, and returns the slice concatenated with the `letters` string. If the `from_front` parameter is `True` the slice is taken from the beginning of the `letters` string and concatenated onto the front of the `letters` string. If the `from_front` parameter is `False` the slice is taken from the end of the `letters` string and concatenated onto the end of the `letters` string.

You can assume that the `how_many` parameter is always less than the length of the `letters` string. For example, executing the following `main()` function with the completed function, prints:

```
1. glaglad
2. gladad
3. shortort
4. lilife
```

```
def main():
   print("1.", add_slice("glad", 3, True))
   print("2.", add_slice("glad", 2, False))
   print("3.", add_slice("short", 3, False))
   print("4.", add_slice("life", 2, True))

def add_slice(letters, how_many, from_front):
```

```
    if from_front:
        return letters[ :how_many] + letters

    return letters + letters[-how_many: ]
```

(5 marks)

Question 2 (15 marks)

a) Assume that the variable, `value`, has been initialised to some integer value. Write a boolean expression which tests if `value` is exactly divisible by 13.

```
value % 13 == 0
```

(3 marks)

b) Give the output produced by the following code.

```
condition1 = True
condition2 = False

if condition1 and condition2:
   print("A")
elif not condition1 or condition2:
   print("B")
else:
   print("C")
```

```
C
```

(3 marks)

c) The following program contains five completed functions and an incomplete `main()` function. The completed program does the following:

- gets two random numbers,
- generates a string containing the sum of the two random numbers,
- asks the user for the answer to this sum,
- checks the user's answer,

and

- gives appropriate feedback based on the user's answer.

Below are three example outputs using the completed program (the user input is shown in a larger font):

```
9 + 2 = 11
Excellent!
```

```
5 + 8 = 10
The correct answer: 5 + 8 = 13
```

```
4 + 3 = 5
The correct answer: 4 + 3 = 7
```

Complete the `main()` function below by writing **FOUR statements where each statement contains a call** to one of the five functions. Your completed program should behave in the manner described above.

```
import random
def get_random_number():
   return random.randrange(2, 10)

def get_sum_string(number1, number2):
   sum_str = str(number1) + " + " + str(number2) + " = "
   return sum_str

def get_user_answer(prompt):
   user_input = input(prompt)
   return int(user_input)

def get_correct_sum(number1, number2):
   return number1 + number2

def display_results(sum_str, correct_result, user_result):
   if correct_result == user_result:
       print("Excellent!")
   else:
       print("The correct answer:", sum_str + str(correct_result))

def main():
    num1 = get_random_number()
    num2 = get_random_number()


    sum_string = get_sum_string(num1, num2)

    correct_answer = get_correct_sum(num1, num2)

    user_answer = get_user_number(sum_string)

    display_results(sum_string, correct_answer,
                                          user_answer)


```

(9 marks)

```
main()
```

Question 3 (15 marks)

a) Give the output produced by the following code.

```
total = 10
for counter in range(0, 12, 4):
   if total > 6:
       total = total - counter
   else:
       total = total + counter
   print(counter, total)

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

```
0 10
4 6
8 14
total: 14
```

(3 marks)

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

```
total = 10

for num in range(15, 0, -3):
   total = total + num
   print(total)

print("End")
```

```
total = 10
num = 15

while num > 0:
   total = total + num
   print(total)
   num = num - 3

print("End")
```

(3 marks)

c) Complete the `get_random_digits()` function which returns a string made up of a series of random digits: 1, 2, 3, 4, 5, 6, 7, 8 or 9. The function is passed one parameter, an integer which represents the number of random digits in the string which is to be returned. For example, executing the following program with the completed function, may print:

```
1. 66947
2. 519
3. 2245132714
```

```
import random

def main():
   print("1.", get_random_digits(5))
   print("2.", get_random_digits(3))
   print("3.", get_random_digits(10))

def get_random_digits(number_of_digits):
```

```
    random_digits = ""

    for num in range(number_of_digits):
        digit = random.randrange(1, 10)
        random_digits = random_digits +
                                    str(digit)

    return random_digits
```

(3 marks)

```
main()
```

d) Complete the output produced by the following program.

```
def muddle1(list1, list2):
   list3 = list1
   list1 = list2
   list2 = list3

def main():
   list1 = [1, 2, 3]
   list2 = [4, 5, 6]
   muddle1(list1, list2)
   print("list1:", list1)
   print("list2:", list2)

main()
```

```
list1: [1, 2, 3]
list2: [4, 5, 6]
```

(3 marks)

e) Complete the output produced by the following program.

```
def muddle2(tuple1, tuple2):
   tuple1 = tuple1 + (tuple2[0],)
   tuple2 = tuple2 + (tuple1[0],)
   a_list = list(tuple2 + tuple1)
   a_list.pop()
   return tuple(a_list)

def main():
   tuple1 = (1, 2)
   tuple2 = (3, 4)
   tuple1 = muddle2(tuple1, tuple2)
   print("tuple1:", tuple1)
   print("tuple2:", tuple2)

main()
```

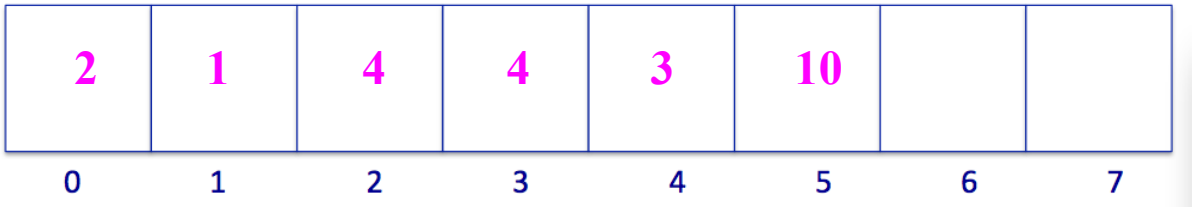
```
tuple1: (3, 4, 1, 1, 2)
tuple2: (3, 4)
```

(3 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, 2, 4, 3, 1]

a_list.insert(3, 4)
value = a_list.pop()
a_list.insert(2, value)
value = a_list.pop(0)
value = value * a_list.index(4)
a_list.append(value)
```



(3 marks)

b) Give the output produced by the following code.

```
a_list = [2, 6, 1]
for index in range(len(a_list)):
   position = a_list[index]
   if position < len(a_list):
       a_list[position] = a_list[position] + 1
       print(index, a_list)
```

```
0 [2, 6, 2]
2 [2, 6, 3]
```

(3 marks)

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

```
a_list = [True, 4, "7", '56']

object1 = a_list[2] * 3
object2 = a_list[0:1]
object3 = a_list[3].rfind("4")
```

```
object1 is of type: str

object2 is of type: list

object3 is of type:  int
```

(3 marks)

d) Complete the `convert_first_letter()` function which is passed a list of names as a parameter. The function changes the first letter of each name in the list to uppercase, leaving the rest of the name unchanged. You can assume that each element of the list contains at least one character. For example, executing the following program with the completed function, prints:

```
1. names: ['karl', 'Orlando', 'carlo', 'zAC']
2. names: ['Karl', 'Orlando', 'Carlo', 'ZAC']
```

NOTE: you can assume that all the elements in the list contain two or more letters.

```
def main():
   names = ["karl", "Orlando", "carlo", "zAC"]
   print("1. names:", names)
   convert_first_letter(names)
   print("2. names:", names)

def convert_first_letter(names_list):
```

```
    for index in range(len(names_list)):
        name = names_list[index]
        name = name[0].upper() + name[1:]
        names_list[index] = name
```

(6 marks)

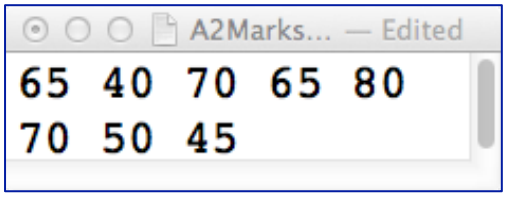
```
main()
```

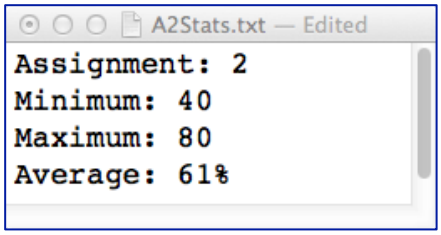
Question 5 (15 marks)

The following program reads information from the input file, `"A2Marks.txt"`, and writes a summary of the information to the output file, `"A2Stats.txt"`.

- The input file contains a series of marks out of 100. The marks are all whole numbers and are separated by a blank space.
- The output file displays four lines of text summarising the assignment marks.

Below is an example of an `"A2Marks.txt"` file (on the left) and the corresponding `"A2Stats.txt"` file (on the right) produced by the completed program:





a) Complete the `get_list_of_marks()` function which is passed one parameter, the name of a file which contains whole numbers (the marks) separated by blank spaces. This function returns a list of all the numbers in the file. Each element of the returned list **should be an integer value**.

b) Complete the `get_marks_stats_tuple()` function which has two parameters: the assignment number and a list of marks. The function returns a tuple made up of four integer values: the assignment number, the minimum mark, the maximum mark and the average mark rounded to the nearest whole number. You can assume that the list contains at least one element and that all the elements of the list are integers.

c) Complete the `write_stats_to_file()` function which has two parameters: the name of the file and a tuple containing four integers. This function writes four lines of text to the file (see the example output file above on the right) displaying:

the assignment number,
the minimum mark,
the maximum mark, and,
the average mark.

```
def main():
    list_of_marks = get_list_of_marks("A2Marks.txt")
    marks_stats_tuple = get_marks_stats_tuple(2, list_of_marks)
    write_stats_to_file("A2Stats.txt", marks_stats_tuple)

def get_list_of_marks(filename):
```

```
file_in = open(filename, "r")
contents = file_in.read()
file_in.close()
```

```
    list_of_marks = contents.split()
    for index in range(len(list_of_marks)):
        list_of_marks[index] =
                        int(list_of_marks[index])

    return list_of_marks
```

```
def get_marks_stats_tuple(assignment_num, marks_list):
```

```
    minimum_mark = min(marks_list)
    maximum_mark = max(marks_list)
    average = round(sum(marks_list) /
                                len(marks_list))
    return (assignment_num, minimum_mark,
                          maximum_mark, average)
```

```
def write_stats_to_file(filename, stats_tuple):
```

```
    assignment_num = stats_tuple[0]
    minimum_mark = stats_tuple[1]
    maximum_mark = stats_tuple[2]
    average = stats_tuple[3]

    file_out = open(filename, "w")
    file_out.write("Assignment: " +
                    str(assignment_num) + "\n")
    file_out.write("Minimum: " +
                      str(minimum_mark) + "\n")
    file_out.write("Maximum: " +
                      str(maximum_mark) + "\n")
    file_out.write("Average: " +
                           str(average) + "%\n")
    file_out.close()
```

```
main()
```

(15 marks)

Question 6 (15 marks)

a) Complete the following code which adds 1 to all the values corresponding to the keys in the `a_dict` dictionary. For example, the output of the completed code is:

```
1. {'c': 4, 'a': 3, 'b': 6}
2. {'c': 5, 'a': 4, 'b': 7}
```

```
a_dict = {"a": 3, "b": 6 , "c": 4}
print("1.", a_dict)

for key in a_dict:
    a_dict[key] = a_dict[key] + 1

print("2.", a_dict)
```

(4 marks)

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

```
def main():
    words=["exceptional","big","delectable","and","peculiar"]
    phrase = ""
    for a_word in words:
        phrase = phrase + get_a_word(a_word, 2) + " "

    print("phrase:", phrase.strip())

def get_a_word(word, index):
    a_dict = {
        "delectable":["appetizing","luscious","yummy","tasty"],
        "exceptional": ["extraordinary", "grand", "rare", "fine"],
        "large": ["great", "huge", "big"],
        "and": ["along with", "plus", "too", "also"],
        "peculiar": ["strange", "unusual", "odd", "funny"]
        }

    if word in a_dict:
        alternatives = a_dict[word]
        return alternatives[index]

    return word
```

```
phrase: rare big yummy too odd
```

(4 marks)

c) Complete the `get_first_last_dict()` function which is passed a list of words as a parameter. The function returns a dictionary where:

- the keys are strings made up of the first letter concatenated with the last letter of a word,
- the corresponding values are a list of all the words where the first letter concatenated with the last letter is the key.

NOTE: any word in the list which has fewer than two characters is ignored.

Executing the following program with the completed function, prints:

```
dict:{'st':['soft','seat'],'le':['love'],'my':['may','many']}
```

```
def main():
    list_of_words = ["may", "a", "many", "soft", "seat", "love"]
    a_dict = get_first_last_dict(list_of_words)
    print("dict:", a_dict)

def get_first_last_dict(word_list):
```

```
a_dict = {}
for word in word_list:
    if len(word) > 1:
        first_last = word[0] + word[-1]
        if first_last in a_dict:
            a_dict[first_last].append(word)
        else:
            a_dict[first_last] = [word]

return a_dict
```

(7 marks)

```
main()
```

Question 7 (10 marks)

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

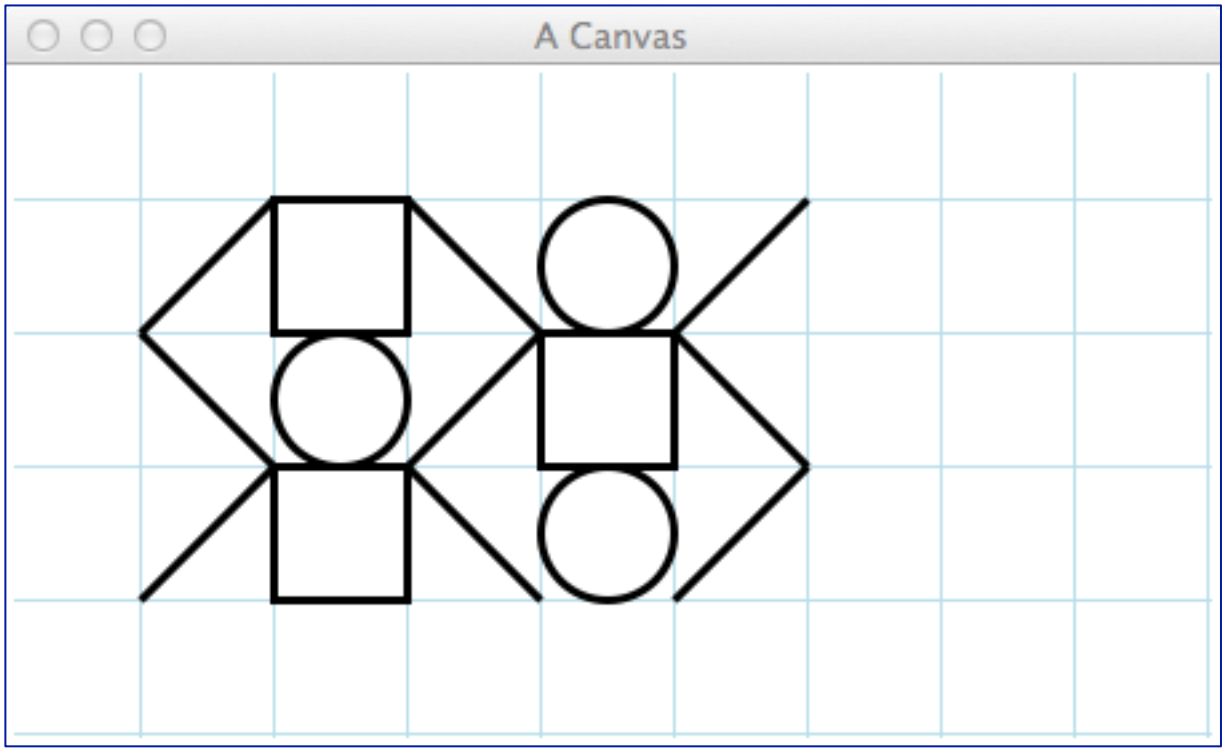
```
from tkinter import *

def draw_pattern(a_canvas):
   symbols_line1 = "1S2O1"
   symbols_line2 = "2O1S2"

   size = 10
   start_left = size
   down = size
   number_to_do = 3
   current_line = symbols_line1

   while number_to_do > 0:
       left = start_left
       for symbol in current_line:
           area = (left, down, left + size, down + size)
           if symbol == "1":
                a_canvas.create_line(left, down + size,
                                                left + size, down)
           elif symbol == "2":
                a_canvas.create_line(left, down, left + size,
                                                      down + size)
           elif symbol == "S":
                a_canvas.create_rectangle(area)
           elif symbol == "O":
                a_canvas.create_oval(area)
           left =  left + size

       down = down + size
       number_to_do = number_to_do - 1
       if current_line == symbols_line1:
           current_line = symbols_line2
       else:
           current_line = symbols_line1

def main():
   root = Tk()
   root.title("A Canvas")
   root.geometry("455x255+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, the variable `current_line`, is a string of symbols. What kind of shape corresponds to the character `'S'` in this variable?

A rectangle

(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)