THE UNIVERSITY OF AUCKLAND

SUMMER SEMESTER, 2017
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	
Preferred Name (if different to forenames)	
Student ID	
Username	

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

ID: ....

Question 1 (15 marks)

a) Complete the output produced by the following code.

```
a = 15 // 4 + 5 % 4
b = 4 % 10 + 5 / 2
print("a:", a, "b:", b)
```

```
a:    4          b:   6.5
```

(2 marks)

b) Complete the output produced by the following code.

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

```
Result: 8
```

(2 marks)

c) In the `main()` function below, complete the 2 calls to the `print_result()` function so that the output of the program is:

```
A.
C.
```

```
def print_result(condition1, condition2, condition3):
     if condition1 and condition2:
          print("A.")
     elif not condition1 and condition3:
          print("B.")
     elif not condition1 or condition3:
          print("C.")

def main():
    print_result(True, True, False) #3rd can be True

    print_result( True, False, True)
```

(4 marks)

```
main()
```

d) The following program, when completed, prints the output shown below. Complete the `main()` function by writing **TWO statements where each statement contains a call** to one of the two functions defined in the program (`print_ticket_prices()` and `get_ticket_prices()`).

```
Child $12
Concession $18
Adult $25
```

```
def print_ticket_prices(prices):
   descriptions_list = ['Child', 'Concession', "Adult"]
   for index in range(len(descriptions_list)):
      print(descriptions_list[index] + " $" + str(prices[index]))

def get_ticket_prices(index):
   list_of_options = [(10, 18, 23), (12, 18, 25), (14, 20, 29)]
   return list_of_options[index]

def main():
   prices = get_ticket_prices(1)
   print_ticket_prices(prices)
                                                    (5 marks)
main()
```

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

```
def do_a_check(value1, value2, value3):
   """Checks the parameter values
   #first two parameters must be the smallest
                                and the second smallest
   >>> do_a_check(3, 4, 1)
   False
                                                    (2 marks)
   """
   a_list = [value1, value2, value3]
   a_list.sort()

   return a_list[0] == value1 and a_list[1] == value2

import doctest
doctest.testmod()
```

ID: ………………

Question 2 (15 marks)

a) Complete the `increase_front_number()` function which is passed two parameters: a string and an integer. The string parameter always starts with exactly three digits followed by one or more characters. The function adds the integer parameter to the number formed by the first three digits of the parameter string. This new number will always be a number with one, two or three digits and you can assume that it will never be a four digit number. The function returns a string made up of a new number with exactly three digits (padded with zeroes if needed) followed by the characters of the parameter string starting from index three. For example, executing the following program with the completed function prints:

```
030tbon
009sli
022ntur
```

```
def main():
   print(increase_front_number('026tbon', 4))
   print(increase_front_number('006sli', 3))
   print(increase_front_number('015ntur', 7))

def increase_front_number(user_name, increase_amt):
```

```
last_part = user_name[3:]
number = int(user_name[:3])
number = number + increase_amt
if number < 10:
   number = "00" + str(number)
elif number < 100:
   number = "0" + str(number)
else:
   number = str(number)

return number + last_part
```

(9 marks)

```
main()
```

ID: ………………

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

```
def main():
   list1 = [3, 2, 1]
   fiddle1(list1)
   print("List1:", list1)

def fiddle1(list1):
   list2 = list1
   list1.append(4)
   for index in range(len(list1)):
      list2[index] = list2[index] + 2
```

```
list1:  [5, 4, 3, 6]
```

(3 marks)

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

```
def main():
   list1 = [3, 2, 1]
   fiddle2(list1)
   print("List1:", list1)

def fiddle2(list1):
   list2 = []
   list2.append(list1[0])
   list2.append(list1[1])
   list1 = list2
   list1[0] = list2[0] + 3
```

```
list1:  [3, 2, 1]
```

(3 marks)

Question 3 (15 marks)

a) Using a `while` loop, complete the code below so that the word `'hello'` is printed 50 times.

```
count = 0

while count < 50:

        print('Hello')
        count = count + 1
```

(3 marks)

b) Rewrite the following function using an equivalent `for` loop instead of the `while` loop:

```
def print_numbers(start_num):
   num = start_num + 6
   count = 0
   while count <= 10:
      print(num)
      num = num + 6
      count = count + 2
```

```
def print_numbers(start_num):
```

```
num = start_num + 6

for count in range(0, 11, 2):
   print(num)
   num = num + 6
```

(4 marks)

c) Give the output produced by the following code.

```
count_a = 4
count_b = 0

while count_a > 0:
   print(count_a, end = " ")
   if count_a % 2 == 0:
      count_a = count_a + 1
      count_b = count_b + 1
   else:
      count_a = count_a – 3

print()
print("End:", count_a, count_b)
```

```
4 5 2 3
End: 0 2
```

(4 marks)

d) Complete the output produced by the following code. The input entered by the user is shown in bold and in a larger font.

```
maximum = int(input("Budget $"))
item_prices = ""
total = 0
amount = int(input("  Item $"))

while total + amount <= maximum:
   total = total + amount
   item_prices = item_prices + " " + str(amount)
   amount = int(input("  Item $"))

print()
print("(", item_prices.strip(), ") Total $", total, sep = "")
```

```
Budget $50
  Item $25
  Item $15
  Item $20

(25 15) Total $40
```

(4 marks)

ID: ………………

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, 4, 6, 2]
a_list.insert(4, 3)
value = a_list.pop()
a_list.insert(3, value)
value = a_list.pop(1)
value = value * a_list.index(6)
a_list.append(value)
```

5	4	2	6	3	9		
0	1	2	3	4	5	6	7

(3 marks)

b) Give the output produced by the following code.

```
a_list = [3, 0, 6, 3, 2, 4]
total = 0

for index in range(len(a_list)):
   if index > a_list[index]:
      print(index, total)
      total =  total + a_list[index]

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

```
1 0
4 0
5 2
Total: 6
```

(3 marks)

ID: ………………

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

```
a_tuple = (3, 4, "five")
a_list = [2, 'one', 5, 4]
a_dict = {'a': 'bdegh', 'b': '135'}

object1 = a_tuple[2:]
object2 = a_list[1].isalpha()
object3 = [a_list[0]  * a_dict['a']]
```

```
object1 is of type: tuple

object2 is of type: bool (boolean)

object3 is of type: list
```

(3 marks)

d) Complete the `remove_non_legal()` function which is passed three parameters: a list of names, followed by a letter and followed by another letter. The function removes any names from the parameter list which do not contain both the letters passed in as parameters either upper or lower case. For example, executing the following program with the completed function, prints:

```
Names which contain both the letter 'a' and the letter 'e':
 ['Delilah', 'LeilA', 'MAZIE']
```

```
def main():
   names = ['Delilah', 'LeilA', "Baldassar", 'MAZIE', "Fede"]
   remove_non_legal(names, 'e', 'a')
   print("Names which contain both 'a' and 'e':")
   print(names)

def remove_non_legal(names_list, letter1, letter2):
```

```
letter1 = letter1.lower()
letter2 = letter2.lower()
for index in range(len(names_list) - 1,
                                         -1,-1):
   name = names_list[index]
   name = name.lower()
   if not (letter1 in name and
                         letter2 in name):
       names_list.pop(index)
```

(6 marks)

```
main()
```

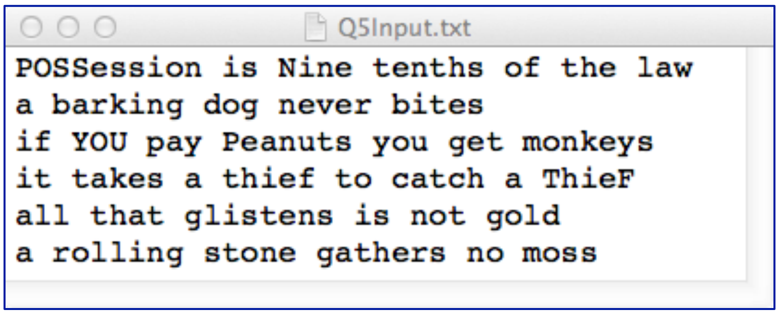
ID: ………………

Question 5 (15 marks)

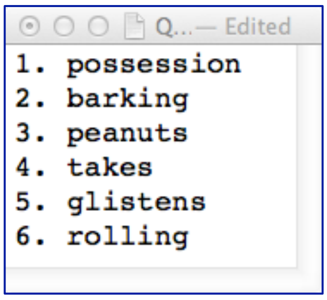
The following program reads information from the input file, " Q5Input.txt", and writes a numbered list of words to the output file, "Q5Output.txt".

- The input file contains English sayings, one saying per line.
- The output file displays a numbered list of the longest word from each saying.

Below is an example of a "Q5Input.txt" file (on the left) and the corresponding "Q5Output.txt" file (on the right) produced by the completed program:



```
POSSession is Nine tenths of the law
a barking dog never bites
if YOU pay Peanuts you get monkeys
it takes a thief to catch a ThieF
all that glistens is not gold
a rolling stone gathers no moss
```



```
1. possession
2. barking
3. peanuts
4. takes
5. glistens
6. rolling
```

a) Complete the `get_list_of_lines()` function which is passed one parameter, the name of a file which contains a series of sayings, one saying per line. This function returns a list of all the sayings. Each element of the returned list should be **in lowercase characters**.

b) Complete the `get_longest_word_in_saying()` function which is passed one parameter: a saying made up of words separated by spaces. The function returns the longest word in the parameter saying. If two words in the saying have the same length the function returns the first word from the start of the saying which has the longest length. For example if the parameter string is `'if you pay peanuts you get monkeys'`, this function returns the word `'peanuts'`.

c) Complete the `write_to_file()` function which has two parameters: the name of a file and a list of words. This function writes a numbered list of all the words in the parameter list to the file (see the example output file above on the right):

```
def main():
    sayings = get_list_of_lines("Q5Input.txt")
    list_of_long_words = get_list_of_longest_words(sayings)
    write_to_file("Q5Output.txt", list_of_long_words)

def get_list_of_lines(filename):
    file_in = open(filename, "r")
    contents = file_in.read()
    file_in.close()
    contents = contents.lower()

    contents_list = contents.split("\n")
    return contents_list
```

ID: ………………

```
def get_list_of_longest_words(list_of_sayings):
    list_of_longest_words = []
    for saying in list_of_sayings:
         longest = get_longest_word_in_saying(saying)
         list_of_longest_words.append(longest)
    return list_of_longest_words

def get_longest_word_in_saying(saying):
    words = phrase.split()
    longest_word = ""
    for word in words:
        if len(word) > len(longest_word):
            longest_word = word
    return longest_word

def write_to_file(filename, list_of_long_words):
    file_out = open(filename, "w")
    number = 1

    for element in list_of_long_words:
        file_out.write(str(number) + ". " +
                                        element +"\n")
        number = number + 1
    file_out.close()

main()
```

(15 marks)

Question 6 (15 marks)

a) Complete the following code which changes all the values corresponding to the keys in the `a_dict` dictionary to uppercase characters. The output of the completed code is:

```
1. {2:'trial', 4:'Bat', 5:'Table', 7:'sensible'}
2. {2:'TRIAL', 4:'BAT', 5:'TABLE', 7:'SENSIBLE'}
```

```
a_dict = {2:'trial', 4:'Bat', 5:'Table', 7:'sensible'}
print("1.", a_dict)

for key in a_dict:
    a_dict[key] = a_dict[key].upper()

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

(4 marks)

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

```
def main():
   keys_str = 'SERENE'
   values = [7, 3, 5, 9]
   a_dict = {}

   for index in range(4):
      letter = keys_str[index]
      score = values[index]
      if letter in a_dict:
          a_dict[letter] = max(score, a_dict[letter])
      else:
          a_dict[letter] = score

   sorted_keys = list(a_dict.keys())
   sorted_keys.sort()
   for key in sorted_keys:
      print(key, "-", a_dict[key])
```

```
E - 9
R - 5
S – 7
```

(4 marks)

c) Complete the `get_most_popular_key()` function which returns the key from the parameter dictionary whose corresponding value has the largest number of elements. For example, executing the following program with the completed function gives the output:

```
Most popular key: B - [65, 73, 65, 78, 65, 76]
```

Note: you can assume that no two values in the dictionary have the same length.

```
def main():
   a_dict = {'A': [],
          'B': [65, 73, 65, 78, 65, 76],
          'C': [55, 51, 60],
          'D': [1, 13, 39, 3]
          }
   popular = get_most_popular_key(a_dict)
   print("Most popular key:", popular, "-", a_dict[popular])

def get_most_popular_key(a_dict):
```

```
most_popular = ""
biggest_so_far = 0

for key in a_dict:
   length = len(a_dict[key])
   if length > biggest_so_far:
       biggest_so_far = length
       most_popular = key

return most_popular
```

(7 marks)

```
main()
```

ID: ………………

Question 7 (10 marks)

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

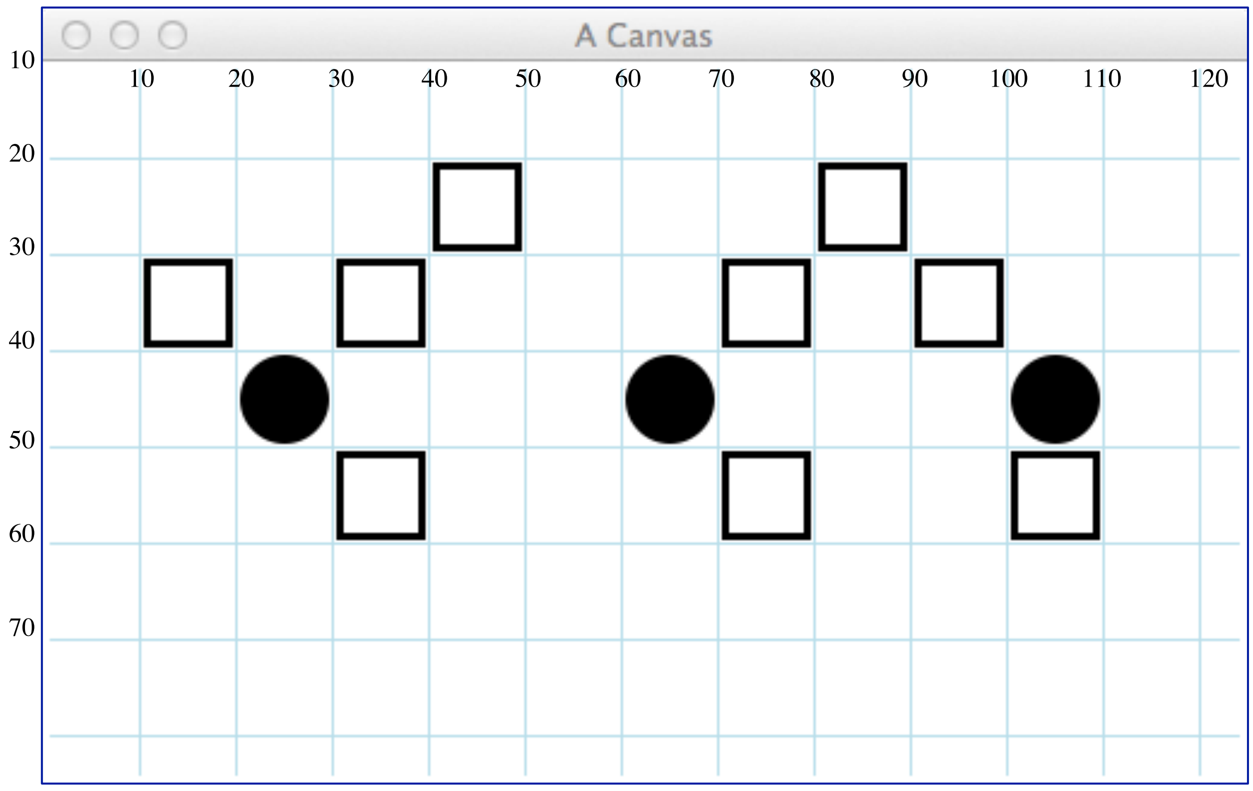
```
from tkinter import *

def draw_pattern(a_canvas):
   size = 10
   start_positions = [size * 4, size, size * 2, size * 3]
   line_fills = ['10001', '101000101', '200020002', '10001001']
   top = size

   for index in range(len(start_positions)):
      left = start_positions[index]
      current_line = line_fills[index]
      for symbol in current_line:
         area = (left, top, left + size, top + size)
         if symbol == "1":
            a_canvas.create_rectangle(area)
         if symbol == "2":
            a_canvas.create_oval(area, fill='black')
         left =  left + size

      top = top + size

def main():
   root = Tk()
   root.title("A Canvas")
   root.geometry("255x155+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 digits. What kind of shape corresponds to the string digit `'2'` in this variable?

circle

(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)