THE UNIVERSITY OF AUCKLAND

SEMESTER TWO, 2019
Campus: City

Computer Science

Principles of Programming

(Time Allowed: TWO hours)

NOTE:
You must answer **all** questions in this test.
No calculators or smart watches 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 (/32)	**Q4** (/12)	**Q7** (/10)
Q2 (/12)	**Q5** (/10)	**TOTAL** (/100)
Q3 (/12)	**Q6** (/12)	

Question 1 (32 marks)

a) Complete the output produced by the following code.

```
result1 = 5 * (2 + 2 ** 3 - 7) - 2 * 5 ** 2 / 10 + 1

result2 = 13 // 4 / 2 + 4 - 12 // 5

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

```
result1:  11.0          result2:   3.5
```

(3 marks)

b) Complete the output produced by the following code.

```
result1 = 22 % 5 + 33 % 6

result2 = 33 % 3 + 6 % 12 - 15 % 15

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

```
result1:  5          result2:   6
```

(3 marks)

c) Complete the output produced by the following code.

```
phrase = "-Too much of a good thing is wonderful-"

position1 = phrase.find("oo")
position2 = phrase.rfind("oo")
little_phrase = phrase[ : position1] + phrase[position2 + 2 : position2 + 6]
little_phrase = little_phrase.strip()
little_phrase = little_phrase.upper()

print("**" + little_phrase + "**")
```

```
**-TD TH**
```

(3 marks)

d) Complete the output produced by the following code.

```
word = "ENERGY"
result = (word[ : : 3] * 2).lower() + word[ : : -2]
print("**" + result + "**")
```

```
**ererYRN**
```

(3 marks)

e) Give the smallest and largest possible number which could be the output of the following code.

```
number1 = random.randrange(5, 25, 5)
number2 = random.randrange(1, 16, 3)
total = number1 + number2
print(total)
```

smallest: 6 largest: 33

(3 marks)

f) Assume that the string, word, has been initialised. Write a boolean expression which tests if the string, word, has more than four letters and ends with the letters "tions".

```
len(word) > 4 and word[-5:] == "tions"
```

(3 marks)

g) In the docstring of the mystery_g() function below, add a short description (fifteen words or less) of the function.

```
def mystery_g(phrase):
    """
    Returns a random word from all the distinct
    words in the parameter string.
    """
```

(3 marks)

```
    words = phrase.split()
    position = random.randrange(len(words))
    return words[position]
```

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

```
a_dict = {"go": [4, 7, 11], "stop": [1, 6, 8, 9], "wait": [2, 3]}

for object_one in a_dict.items():
    object_two = object_one[1]
    object_three = object_two[-1]
```

`object_one` is of type: **tuple**

`object_two` is of type: **list**

`object_three` is of type: **int**

(3 marks)

i) Complete the output produced by the following code.

```
list1 = [6, 2, 5]
list2 = list1
list3 = list2
list3.pop(2)
list2.append(list1[0])

print("1:", list1 == list3, "  2:", list1 is list3, "  3:",
                                              len(list1) == len(list2))
```

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

(2 marks)

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

```
def main():
    a_list = [3, 4, 1]
    fiddle1(a_list)
    print("a_list:", a_list)

def fiddle1(list1):
    elements_to_add = [5, 5, 3]
    list2 = list1
    for element in elements_to_add:
        if element not in list1:
            list2.append(element)
    list1.pop(1)
```

```
a_list: [3, 1, 5]
```

(2 marks)

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

```
def main():
    a_list = [3, 5, 7]
    fiddle2(a_list)
    print("a_list:", a_list)

def fiddle2(list1):
    list2 = list1
    list1 = [3, 4]
    list2.reverse()
```

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

(2 marks)

l) 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

    >>> do_a_check("abc", "xyc123")
    False

   """

    count = 0
    min_len = min(len(value1), len(value2))
    for i in range(min_len):
        if value1[i] == value2[i]:
            count = count + 1

    return count > 2

import doctest
doctest.testmod()
```

(2 marks)

Question 2 (12 marks)

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

```
def main():
    function_ifs(10, 7, 6)

def function_ifs(a, b, c):
  if a > b and a < c:
       print("A", end = " ")
  elif not (c > b or c > a):
       print("B", end = " ")
       if c < b:
           print("C", end = " ")
  else:
       print("D", end = " ")

  if a > b or c > b:
       print("E", end = " ")
  elif not a > b or a + c > b:
       print("F", end = " ")

  print("G", end = " ")
```

```
B C E G
```

(3 marks)

b) What is the output produced by the following code?

```
number = 16

while number > 4:
    number = number - 5

    if number % 2 == 0:
        number = number - 1

    print(number, end = " ")

print("Stop.")
```

```
11 5 -1 Stop.
```

(3 marks)

c) Complete the `get_total()` function which continuously prompts the user to enter numbers until the user enters a zero. The function returns the total of all the numbers entered by the user which are between 1 and 99 (both inclusive). You can assume the user will always enter integers. Below are two possible outputs produced when the following `main()` function is executed using the completed `get_total()` function (the user input is shown in bold).

```
Enter number (0 to stop): 3
Enter number (0 to stop): -55
Enter number (0 to stop): 654
Enter number (0 to stop): 2
Enter number (0 to stop): 1
Enter number (0 to stop): -5
Enter number (0 to stop): 100
Enter number (0 to stop): 0
Total of numbers between 1 and 99: 6
```

```
Enter number (0 to stop): 387
Enter number (0 to stop): -55
Enter number (0 to stop): 232
Enter number (0 to stop): -5
Enter number (0 to stop): 99
Enter number (0 to stop): 0
Total of numbers between 1 and 99: 99
```

```
def main():
    user_total = get_total()
    print("Total of numbers between 1 and 99:", user_total)

def get_total():
    number_prompt = "Enter number= (0 to stop): "

    total = 0
    user_number = int(input(number_prompt))
    while user_number != 0:
        if user_number > 0 and user_number < 100:
            total = total + user_number
        user_number = int(input(number_prompt))

    return total
```

(6 marks)

Question 3 (12 marks)

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

```
def main():
    numbers = [45, 24, 9, 12, 14]
    print("result:", get_result(numbers, 15))

def get_result(all_numbers, number):
    index = -1
    closest_difference = "N/A"
    for i in range(len(all_numbers)):
       difference = all_numbers[i] - number
       if difference > 0:
          if closest_difference == "N/A" or difference < closest_difference:
                closest_difference = difference
                index = i
    return index
```

```
result:  1
```

(3 marks)

b) Complete the assignment statement in the `main()` function below so that the output produced when the `main()` function is executed is: `result: False`

```
def main():
    word =    "abc"  #no repeated letters
```

(3 marks)

```
    result = do_a_check(word)
    print("result:", result)

def do_a_check(word):
    letter_list = []
    for letter in word:
        if letter in letter_list:
            return True
        else:
            letter_list.append(letter)
    return False
```

c) Complete the `has_all_ingredients()` function which is passed two lists of strings as parameters. The function returns `True` if all the elements of the second parameter list, `needed`, are also elements of the first parameter list, `larder`. Otherwise the function returns `False`. Below is the output produced when the following `main()` function is executed using the completed `has_all_ingredients()` function.

```
1. True
2. False
```

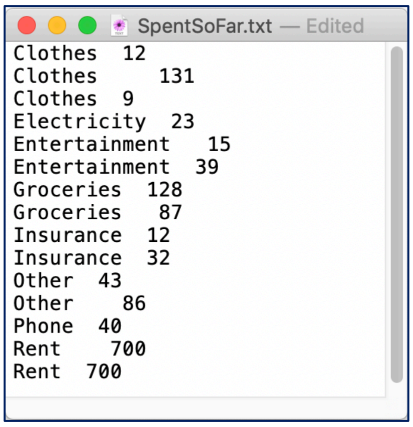
```
def main():
    larder = ["eggs", "spices", "fruit", "flour", "sugar", "butter",
  "oil", "bread", "milk"]
    ingredients_needed = ["eggs", "fruit", "butter", "flour", "milk"]
    print("1.", has_all_ingredients(larder, ingredients_needed))
    print("2.", has_all_ingredients(larder, ["sugar", "walnuts"]))

def has_all_ingredients(larder, needed):
```

```
    for ingredient in needed:
        if ingredient not in larder:
            return False
    return True
```

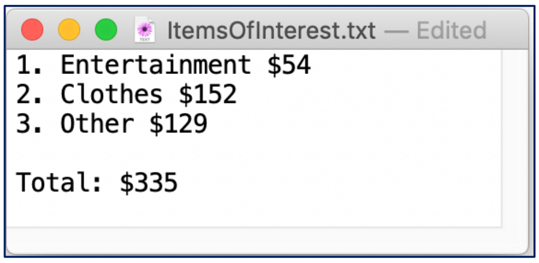
(6 marks)

Question 4 (12 marks)

Complete the three functions in the following program which reads information from the input file, `'SpentSoFar.txt'`, processes the information and writes the resulting information to the output file, `'ItemsOfInterest.txt'`.
Below is an example of a `"SpentSoFar.txt"` file (on the left) and the corresponding `"ItemsOfInterest.txt"` file (on the right) produced by the completed program:



```
Clothes  12
Clothes     131
Clothes  9
Electricity  23
Entertainment   15
Entertainment  39
Groceries  128
Groceries   87
Insurance  12
Insurance  32
Other  43
Other    86
Phone  40
Rent    700
Rent  700
```



```
1. Entertainment $54
2. Clothes $152
3. Other $129

Total: $335
```

a) Complete the `read_from_file()` function which is passed the name of a file as a parameter. Each line of the input file is made up of a category of item followed by one or more spaces and the amount spent on that item. The function returns a list of strings where each element corresponds to one line of the file. No element of the returned list should contain newline characters.

b) Complete the `get_total_amount_spent()` function which is passed two parameters:
- a list of strings where each element is made up of a category of item followed by one or more spaces and the amount spent on that category, e.g., `"Entertainment   15"`, and,
- a string which is the category of an item, e.g., `"Entertainment"`.

This function loops through each element of the parameter list and returns the total (an integer) of all the amounts from the parameter list spent on that particular category of item.

c) Complete the `write_to_file()` function which has three parameters: the output filename, a list of strings (`items_of_interest` - the item categories) and a list of integers (`spent_each_item` - amount spent on each category). The two parameter lists have **the same length**. This function first writes a numbered list (starting from 1) of each category (from the `items_of_interest` parameter) and the total spent on that category (from the `spent_each_item` parameter), where each category is separated from the amount spent on that category by `"  $"`. Then a blank line is written followed by a line containing the string `"Total: $"`, followed by the total of the `spent_each_item` parameter list. See the screenshot of the example output file above on the right.

```
def main():
    all_items = read_from_file("SpentSoFar.txt")
    items_of_interest = ["Entertainment", "Clothes", "Other"]
    amount_spent = []
    for category_name in items_of_interest:
        amount = get_total_amount_spent(all_items, category_name)
        amount_spent.append(amount)

    write_to_file("ItemsOfInterest.txt", items_of_interest,
                                                  amount_spent)
```

```
def read_from_file(filename):
    input_file = open(filename, "r")
    file_text = input_file.read()
    input_file.close()
    return file_text.split("\n")

def get_total_amount_spent(item_list, category):
    total_spent = 0
    for information in item_list:
        if category in information:
            split_info = information.split()
            amount = int(split_info[1])
            total_spent = total_spent + amount

    return total_spent

def write_to_file(filename, items_of_interest, spent_each_item):
    numbering = 1
    output_file = open(filename, 'w')
    for i in range(len(items_of_interest)):
        category = items_of_interest[i]
        amount =  spent_each_item[i]
        output_file.write(str(numbering) + ". " +
             category + " $" + str(amount) + "\n")
        numbering = numbering + 1

    output_file.write("\nTotal: $" +
                     str(sum(spent_each_item)))
    output_file.close()
main()
```

(12 marks)

Question 5 (10 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 = [2, 0, 3, 1]
a_list.insert(3, 1)
a_list.insert(3, 2)
index = a_list.index(3)
a_list.insert(0, index)
a_list.append(a_list[2])
a_list.pop(a_list[2])
```

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

(3 marks)

b) Give the output produced by the following code.

```
list1 = [4, 6, 7, 8, 1]
list2 = [7, 6, 5, 9, 2, 7, 7, 2, 7, 6]

small_len = min(len(list1), len(list2))

for index in range(small_len):
    if list1[index] >= list2[index] - 1 and list1[index] <= list2[index] + 1:
        print(list1[index])
```

```
6
8
1
```

(2 marks)

c) Complete the `get_special_total()` function which is passed two parameters: a list of whole numbers (`numbers`) and a single number (`special_number`). The function totals all the **positive** (i.e., > 0) values in the `numbers` list **until** a value which is greater than the special number is reached or until the end of the list has been reached. The function returns the total. For example, executing the following `main()` function gives the output:

```
1. 7
2. 17
3. 17
4. 103
```

```
def main():
    a_list = [4, -6, 3, -77, 10, 21, 3, 56, -7, 6]
    print("1.", get_special_total(a_list, 5))
    print("2.", get_special_total(a_list, 10))
    print("3.", get_special_total(a_list, 20))
    print("4.", get_special_total(a_list, 70))

def get_special_total(numbers, special_number):
```

```
    if len(numbers) == 0:
        return 0

    index = 0
    total = 0
    finished = False

    while not finished and index < len(numbers):
        element = numbers[index]
        if element > special_number:
            finished = True
        elif element > 0:
            total = total + element
        index = index + 1

    return total
```

(5 marks)

Question 6 (12 marks)

a) Complete the following code which fills the `dict3` dictionary with all the dictionary keys which are keys of both `dict1` and `dict2`. The corresponding value of each `dict3` item should be the maximum of the two values corresponding to the same key in `dict1` and `dict2`. The output of the completed code is:

```
ant - 5
zebra - 4
dog - 5
```

```
dict1 = {"ant": 2, "bat": 2, "zebra": 4, "dog": 3, "cat": 9}
dict2 = {"ant": 5, "bee": 7, "zebra": 2, "dog": 5, "monkey": 7}
dict3 = {}

for key in dict1:
    if key in dict2:
        dict3[key] = max(dict1[key], dict2[key])

for key in dict3:
       print(key, "-", dict3[key])
```

(3 marks)

b) Complete the output of the following code:

```
allowed = ["ho", "oh", "op", "po", "hp", "ph"]
text = "HOP9hop87ho"
a_dict = {}
text = text.lower()

index = 1
while index < len(text):
    letters = text[index - 1] + text[index]
    if letters in allowed:
        if letters in a_dict:
            a_dict[letters] = a_dict[letters] + 1
        else:
            a_dict[letters] = 1

    index = index + 1

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

```
a_dict: {'ho': 3, 'op': 2}
```

(3 marks)

c) Complete the `get_text_value()` function which is passed two parameters, a dictionary and a string of text. The keys of the parameter dictionary are single letters and the corresponding values are integers (the value of the key letter), e.g., `{'b': 5, 'a': 6, 'c': 3}`. The function returns the total valuation (an integer) of the string of text where:

- if the letter from the text is a key of the dictionary then its value is the integer corresponding to the letter in the dictionary,
- any vowel characters from the text which are not in the dictionary are worth 1,
- all other characters are worth 0.

Note: you can assume that all the keys in the dictionary are lowercase characters. You will need to change the text to lowercase before you work out the total value of the text.

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

```
1. BLAS! - 4
2. aeiou dgh - 5
3. zebra crossing! - 14
```

```
def main():
    letter_value_dict = {"z": 2, "c": 2, "f":4, "s":3, "v":8}
    words = "BLAS!"
    print("1.", words, "-", get_text_value(letter_value_dict, words))
    words = 'aeiou dgh'
    print("2.", words, "-", get_text_value(letter_value_dict, words))
    words = "zebra crossing!"
    print("3.", words, "-", get_text_value(letter_value_dict, words))

def get_text_value(special_letters, text):
    vowels = "aeiou"

    total_worth = 0
    text = text.lower()
    for character in text:
        if character in special_letters_dict:
            total_worth +=
                  special_letters_dict[character]
        elif character in vowels:
            total_worth += 1

    return total_worth
```

(6 marks)

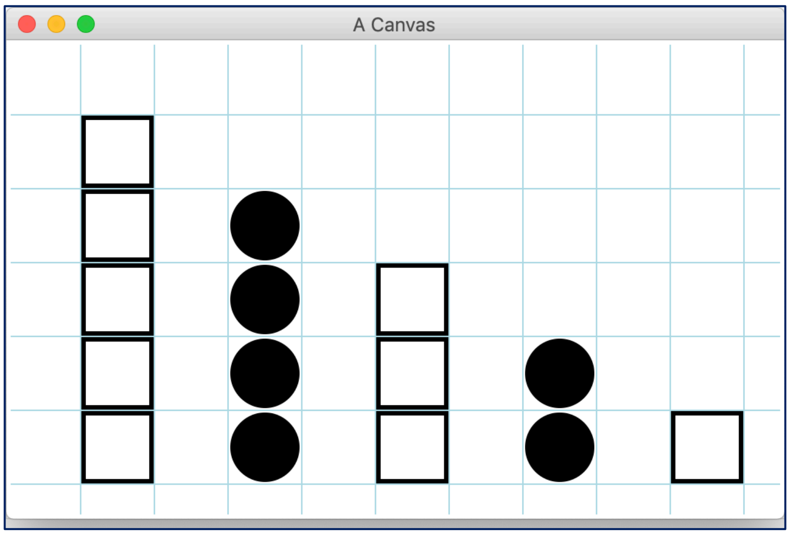
Question 7 (10 marks)

Parts a) and b) of this question refer to two programs which import and use the `tkinter` module. The `main()` functions of the two programs which create the window, create the `Canvas` object and call the functions for Part a) and Part b) of this question are not shown here.

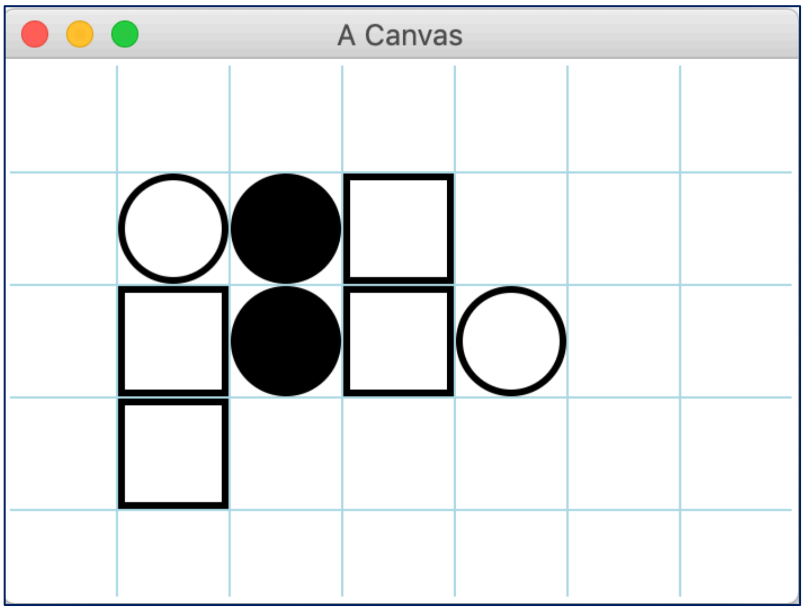
a) In the `draw_pattern()` function below, complete the **four** statements marked #1, #2, #3 and #4 so that the output window is as shown lower down on this page. Grid lines have been drawn in the window to help you. The gap between adjacent gridlines is the same as the variable, `size`, i.e., 10 pixels.

```
def draw_pattern(a_canvas):
    size = 10
    top = size

    for row_number in range( 1, 6 ):                            #1.
        left = size
        is_a_rectangle = True

        for col_number in range( row_number ):                  #2.

            area = (left, top, left + size, top + size)

            if is_a_rectangle:
                a_canvas.create_rectangle(area, width = 3)
            else:
                a_canvas.create_oval(area, fill='black')

            is_a_rectangle = not is_a_rectangle                 #3.

            left = left + size * 2                              #4.

        top = top + size
```

(4 marks)



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



(6 marks)

```
def draw_shapes(a_canvas):
    size = 10
    start_left = size
    top = size

    column_widths = [3, 4, 1]
    type_of_shape = [1, 2, 3, 3, 2, 3, 1, 3, 3, 2, 3, 1, 1, 2]

    current_shape_index = 0

    for row_number in range(len(column_widths)):
        left = start_left

        number_in_row = column_widths[row_number]

        for col in range(number_in_row):
            shape = type_of_shape[current_shape_index]
            current_shape_index = current_shape_index + 1
            area = (left, top, left + size, top + size)
            if shape == 1:
                a_canvas.create_oval(area)
            elif shape == 2:
                a_canvas.create_oval(area, fill="black")
            elif shape == 3:
                a_canvas.create_rectangle(area)

            left = left + size

        top = top + size
```