



ID: .....

**Question 1 (13 marks)**

- a) Complete the output produced by the following code:

```
result = 2 * 7 // 3 - 5 // 3 + 3 % 2
print("Result:", result)
```

Result: 4

(2 marks)

- b) Give the output printed by the following code:

```
subtract = 4
number = 20
for counter in range(1, 10, 3):
    subtract += 1
    print(counter, number, end = " ")
    number = number - subtract
```

1 20 4 15 7 9

(3 marks)

- c) Write a for ... in loop which prints the following numbers:

43 33 23 13 3 -7 -17

```
for num in range(43, -18, -10):
    print(num, end = " ")
```

(3 marks)

ID: .....

- d) Complete the following program which continuously prompts the user for a positive odd number 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, the function then returns the number. Below are two example outputs produced by the completed program. The user input is shown in a bigger font and in bold.

```
Enter a positive odd number: -4
Enter a positive odd number: -5
Enter a positive odd number: 6
Enter a positive odd number: 7
User number: 7
```

```
Enter a positive odd number: 28
Enter a positive odd number: -13
Enter a positive odd number: 11
User number: 11
```

```
def get_legal_number(prompt):
```

```
    user_num = int(input(prompt))

    while user_num % 2 != 1 or user_num < 0:
        user_num = int(input(prompt))

    return user_num
```

(5 marks)

```
def main():
```

```
    number = get_legal_number("Enter a positive odd number: ")
    print("User number:", number)
```

```
main()
```

ID: .....

**Question 2 (10 marks)**

- a) The following function takes two string parameters and returns an integer value. Work out what the function does and rewrite it using a sensible function name and sensible variable names:

```
def blah(a, b):
    c = 0
    for z in range(len(a)):
        d = a[z]
        e = b[z]
        if d == e:
            c += 1
    return c
```

```
def count_matches2(phrase1, phrase2):
    count = 0
    for index in range(len(phrase1)):
        letter1 = phrase1[index]
        letter2 = phrase2[index]
        if letter1 == letter2:
            count += 1
    return count
```

(4 marks)

- b) On the next page complete the `main()` function of the program. The program first displays three tuples, then prints a blank line and finally displays the three tuples with all the duplicate values removed. A value is a duplicate if it occurs more than once in any of the three tuples. All the functions for this program have already been defined. You are required to complete the `main()` function of the program by adding SIX lines of code, with each line making a call to one of the three defined functions. To the right is the output produced by the completed program.

```
1. (1, 2, 3, 7)
2. (4, 1, 6, 8)
3. (5, 8)
```

```
1. (2, 3, 7)
2. (4, 6)
3. (5,)
```

ID: .....

```
def display_tuples(t1, t2, t3):
    print("1.", t1)
    print("2.", t2)
    print("3.", t3)

def remove_duplicates(tup, duplicates):
    a_list = []
    for num in tup:
        if num not in duplicates:
            a_list.append(num)

    return tuple(a_list)

def get_all_duplicate_numbers(t1, t2, t3):
    big_tuple = tuple(t1 + t2 + t3)
    duplicates = []
    elements_so_far = []

    for value in big_tuple:
        if value in elements_so_far:
            duplicates.append(value)
            elements_so_far.append(value)
    return duplicates

def main():
```

```
    t1 = (1, 2, 3, 7)
    t2 = (4, 1, 6, 8)
    t3 = (5, 8)

    display_tuples(t1, t2, t3)

    print()
    duplicates = get_duplicate_elements(t1,t2,t3)

    t1 = remove_duplicates(t1, duplicates)
    t2 = remove_duplicates(t2, duplicates)
    t3 = remove_duplicates(t3, duplicates)

    display_tuples(t1, t2, t3)
```

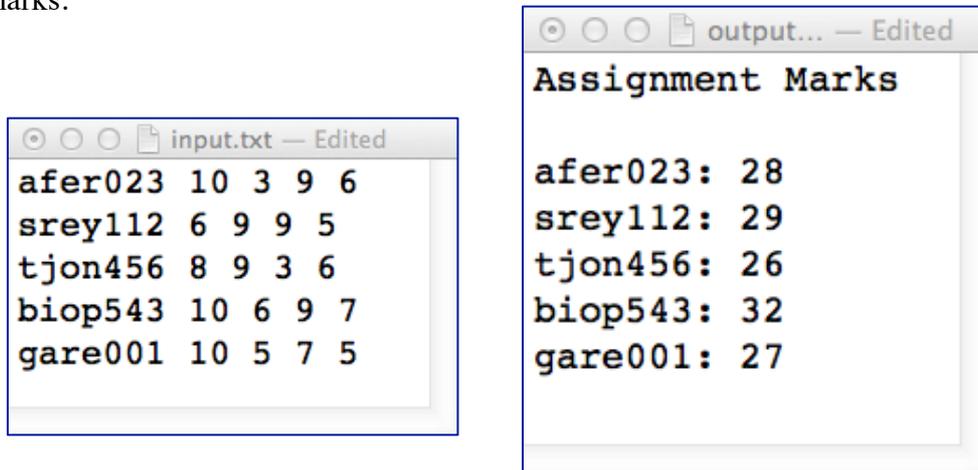
```
main()
```

(6 marks)

ID: .....

**Question 3 (15 marks)**

Complete the three functions in the following program which reads information from the input file, 'input.txt', processes the information and writes the resulting list of information to the output file, 'output.txt'. Each line of the input file is made up of a username followed by 4 assignment marks.



- Complete the `get_list_of_lines()` function which takes a filename as a parameter and returns a list of each line read from the file. None of the string elements of the returned list should contain the newline character.
- Complete the `write_to_file()` function which takes a filename and a list of strings as parameters. The first line written to the file is "Assignment Marks", followed by a blank line. Then each element of the parameter list of strings is written to the file. Each string is written on a new line.
- Complete the `get_username_total_str()` function which has one string parameter. The parameter is a string containing a username followed by numbers separated by spaces, e.g., "afer023 10 3 9 6". The function pulls out the username from the string, totals the four numbers and returns a string made up of the username, followed by a " : ", followed by the total of the four numbers, e.g., "afer023: 28".

```
def main():
    names_marks_list = get_list_of_lines("input.txt")
    output_list = []

    for line_str in names_marks_list:
        username_total_str = get_username_total_str(line_str)
        output_list.append(username_total_str)

    write_to_file("output.txt", output_list)
```

ID: .....

```
def get_list_of_lines(filename):
```

```
    file_in = open(filename, "r")
    contents = file_in.read()
    file_in.close()
    contents_list = contents.split("\n")

    return contents_list
```

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

```
    file_out = open(filename, "w")
    file_out.write("Assignment Marks\n\n")

    for element in info_list:
        file_out.write(element + "\n")

    file_out.close()
```

```
def get_username_total_str(line_str):
```

```
    info_list = line_str.split()
    name = info_list[0]
    info_list = info_list[1:]

    total = 0
    for num_str in info_list:
        total = total + int(num_str)

    name_total_str = name + ": " + str(total)

    return name_total_str
```

```
main()
```

(15 marks)

ID: .....

**Question 4 (10 marks)**

a) Complete the call to the function below so that the output is:

```
pr****nt
```

```
def print_something(text, max_num):  
    if len(text) < 2 or max_num < 4 or max_num < len(text):  
        return text  
  
    num_to_do = max_num - len(text)  
    between = "*" * num_to_do  
    starter = text[0:2]  
    end = text[-2:]  
  
    final_output = starter + between + end  
    print(final_output)
```

```
print_something("present", 12)
```



(3 marks)

```
The string has to have length 7  
and has to start with 'pr' and  
end with 'nt'.
```

ID: .....

- b) Complete the `get_middle_name()` function which is passed one parameter: a string in which there are exactly two spaces. The function returns the middle name, i.e., the name which starts after the first space and ends before the second space in the parameter string. The middle name which is returned should not contain any spaces. Executing the following program with the completed function, prints:

Alexander

Jose

Tao

```
def main():  
    print(get_middle_name("John Alexander Lang"))  
    print(get_middle_name("Jacqueline Jose Smith-Jones"))  
    print(get_middle_name("Lyn Tao Kim"))
```

```
def get_middle_name(full_name):
```

```
    first_space = full_name.find(" ")  
    second_space = full_name.rfind(" ")  
  
    middle_name =  
        full_name[first_space+1:second_space]  
  
    return middle_name
```

```
main()
```

(7 marks)

ID: .....

**Question 5 (16 marks)**

- a) In the boxes below, show each element of `list2` after the following code has been executed. Use as many of the boxes as you need.

```
list1 = [1, 3, 5]
list2 = [2]

list2 = list2 + list1 + [list1[2] + list1[1] + list1[0]]
list1.append(list2[0])
list2.append(list1[0])
list1.pop()
list2.insert(3, 6)
list2.pop(2)
```

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

(5 marks)

- b) Complete the output produced by the following program:

```
def print_funny_sum(a_list):
    total = 0
    numbers_so_far = []

    for num in a_list:
        if num not in numbers_so_far:
            total = total + num
            numbers_so_far.append(num)

    print("Numbers in list:", numbers_so_far)
    print("Funny sum:", total)

def main():
    print_funny_sum([3, 2, 3, 2, 1])

main()
```

ID: .....

```
Numbers in list: [3, 2, 1]
```

```
Funny sum: 6
```

(5 marks)

- c) Complete the `get_words_from_list()` function which is passed two parameters: a list of words and a sentence (a string). The function returns a unique list of all the words from the sentence which are also in the parameter list of words. A word in the sentence only matches a word in the list if they are a completely separate word, e.g., "as" does not match the word "feast". You can assume that all the words are in lower case and that the sentence does not contain any punctuation. Executing the following program with the completed function, prints:

```
List words: ['fall', 'the', 'mind']
```

```
def main():
    phrase = "all our words are but crumbs that fall down from
              the feast of the mind"
    word_list = ["the", "need", "worm", "mind", "fall", "up", "as"]
    print("List words:", get_words_from_list(word_list, phrase))
```

```
def get_words_from_list(a_list, sentence):
```

```
    found_so_far = []
    words_from_phrase = sentence.split()

    for word in words_from_phrase:
        if word in a_list and
            not word in found_so_far:
            found_so_far.append(word)

    return found_so_far
```

(6 marks)

```
main()
```

ID: .....

**Question 6 (14 marks)**

a) Given the following code:

```
object1 = {'a': [2, 3, 1], 'e': [2, 7, 1], 'b': [1, 2, 4] }
object2 = object1["e"]
value = object1["b"]
object3 = value[1]
```

what is the type of the three Python objects: object1, object2 and object3?

```
object1 is of type: dict
object2 is of type: list
object3 is of type: int
```

(3 marks)

b) Complete the output produced by the following program:

```
def main():
    a_dict = {'a':'trs', 'x':'ye', 'p':'eopts', 'e':'aetsrnm',
              'b':'laoeiub'}
    print("Dictionary:", a_dict)
    letter_list = []
    for letter in a_dict:
        if letter in a_dict[letter]:
            letter_list.append(letter)
    print("Letter list:", letter_list)

main()
```

```
Dictionary: {'p':'eopts', 'a':'trs', 'e':'aetsrnm', 'x':'ye',
             'b':'laoeiub'}
Letter list: ['b', 'p', 'e']
```

(5 marks)

ID: .....

- c) Complete the `get_reversed_dict()` function which is passed a Python dictionary as a parameter. The function returns a new dictionary object which is made up of key:value pairs where :

the keys are the values from the key:value pairs of the parameter dictionary,  
and,

the values are the keys from the key:value pairs of the parameter dictionary.

When filling the new dictionary object with the key:value pairs, if the key is already in the new dictionary, then the value is concatenated to the string value which already exists for the key in the new dictionary. Executing the following program with the completed function, prints:

```
dict:          {'e': 4, 'd': 9, 'a': 4, 'b': 9, 'c': 4}
reversed dict: {9: 'db', 4: 'eac'}
```

```
def main():
    dict1 = {'a': 4, 'b': 9, 'c': 4, 'd': 9, 'e': 4}
    dict2 = get_reversed_dict(dict1)
    print("dict:          ", dict1)
    print("reversed dict:", dict2)
```

```
def get_reversed_dict(a_dict):
```

```
    dict2 = {}

    for keyword in a_dict:
        value = a_dict[keyword]

        if value in dict2:
            dict2[value] += keyword
        else:
            dict2[value] = keyword

    return dict2
```

```
main()
```

(6 marks)

ID: .....

**Question 7 (10 marks)**

As accurately as possible, in the window on the next page, show what is drawn by the following program. Grid lines have been drawn in the window to help you. The gap between adjacent gridlines is 10 pixels.

```
from tkinter import *
def draw_pattern(a_canvas):
    size = 10
    start_left = size
    down = size
    starts_with_square = True
    number_across = 3

    while number_across > 0:
        left = start_left
        for num_to_do in range(number_across):
            rect_box = (left, down, left + size, down + size)
            if starts_with_square:
                a_canvas.create_rectangle(rect_box)
            else:
                a_canvas.create_oval(rect_box)
            left = left + size

        left = left + size

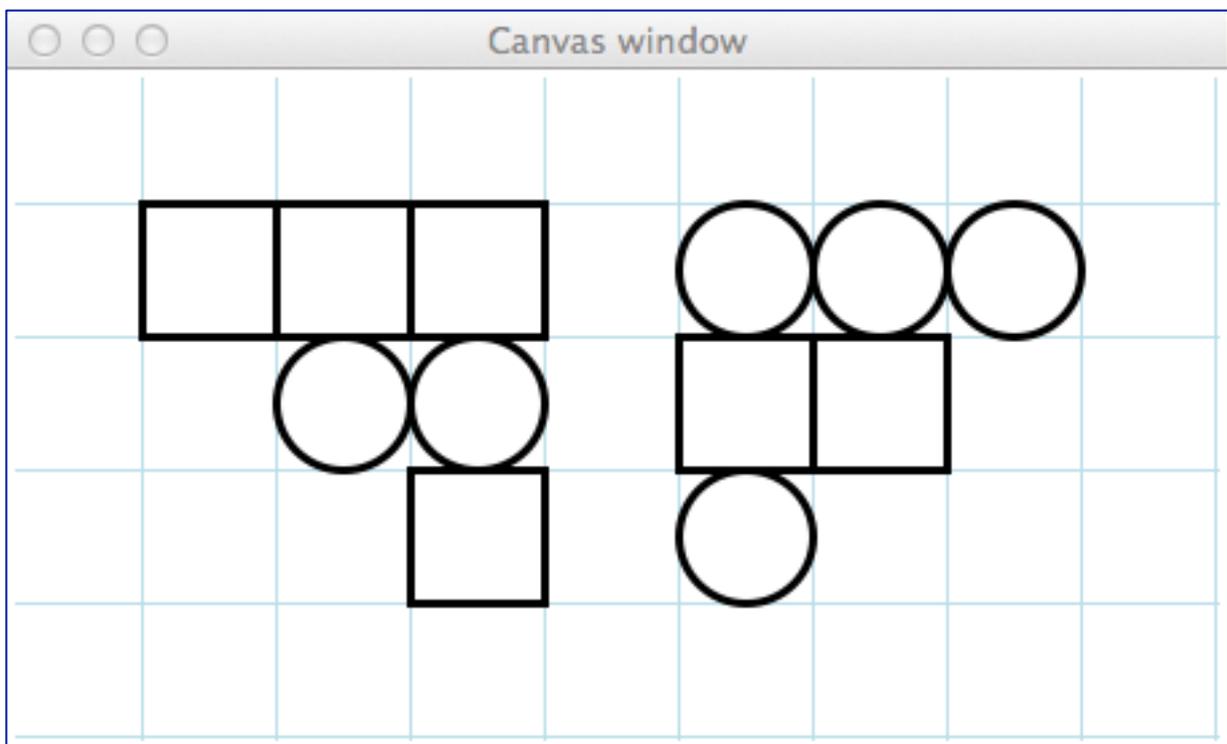
        for num_to_do in range(number_across):
            rect_box = (left, down, left + size, down + size)
            if starts_with_square:
                a_canvas.create_oval(rect_box)
            else:
                a_canvas.create_rectangle(rect_box)
            left = left + size

        down = down + size
        start_left = start_left + size
        number_across = number_across - 1
        starts_with_square = not starts_with_square
```

ID: .....

```
def main():  
    root = Tk()  
    root.title("A Canvas")  
    root.geometry("300x300+10+10")  
    a_canvas = Canvas(root, bg="white")  
    a_canvas.pack(fill=BOTH, expand=1) #Canvas fills the window  
    draw_pattern(a_canvas)  
    root.mainloop()
```

```
main()
```



(10 marks)

ID: .....

**Question 8 (6 marks)**

a) Complete the output produced by the following program:

```
def fiddle_with_lists1(list1, list2):
    for index in range(len(list1)):
        element = list1[index]
        if element in list2:
            list1[index] = list1[index] * 2
    return list1

def main():
    list1 = [1, 2, 3]
    list2 = [1, 5, 4, 2]
    list2 = fiddle_with_lists1(list1, list2)
    print("List1:", list1)
    print("List2:", list2)
    print()

main()
```

```
List1: [2, 4, 3]
```

```
List2: [2, 4, 3]
```

(3 marks)

ID: .....

b) Complete the output produced by the following program:

```
def fiddle_with_lists2(list1, list2):
    list2 = list1
    for index in range(len(list1)):
        element = list1[index]
        if element in list2:
            list1[index] = list1[index] * 2

def main():
    list1 = [1, 2, 3]
    list2 = [1, 5, 4, 2]
    fiddle_with_lists2(list1, list2)
    print("List1:", list1)
    print("List2:", list2)

main()
```

```
List1: [2, 4, 6]
List2: [1, 5, 4, 2]
```

(3 marks)

ID: .....

**Question 9 (6 marks)**

The following function contains a docstring. In the docstring add **two** doctests for the function. Your two tests should have different outcomes and neither of them should fail.

```
def do_a_check(a_list, value):  
    """ Processes a list of numbers
```

```
>>> do_a_check([3, 5, 6, 15, 9], 5)  
2  
>>> do_a_check([32, 8, 56, 12, 24], 4)  
5
```

(6 marks)

```
    """
```

```
    count = 0
```

```
    for num in a_list:
```

```
        if num % value == 0:
```

```
            count += 1
```

```
    return count
```

```
import doctest
```

```
doctest.testmod()
```