

# THE UNIVERSITY OF AUCKLAND

---

**SUMMER SEMESTER, 2015**  
**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.

<b>Surname</b>	
<b>Forenames</b>	
<b>Student ID</b>	
<b>Login (UPI)</b>	

<b>Q1</b>	<b>Q4</b>	<b>Q7</b>
(/12)	(/13)	(/9)
<b>Q2</b>	<b>Q5</b>	<b>Q8</b>
(/15)	(/15)	(/6)
<b>Q3</b>	<b>Q6</b>	<b>TOTAL</b>
(/16)	(/14)	(/100)

ID: .....

**Question 1 (12 marks)**

a) Complete the output produced by the following code.

```
num1 = 5
num2 = 4
num3 = 2
result = (num1 ** num3 - num1 // 2) / 2 - (num1 + num2)
print("Result:", result)
```

Result:

(2 marks)

b) Complete the following for ... in loop so that the word, "bellissimo", is printed twenty times.

```
for
    print("bellissimo")
```

(2 marks)

c) Assume that the variables num1 and num2 have been initialised. Write a Python boolean expression which evaluates to True if num2 is exactly 4 more than num1 or if num2 is exactly 3 less than num1.

(2 marks)

d) Give the output produced by the following code:

```
is_available = False
is_available = is_available or 4 < 6
is_available = not is_available
print(is_available)
```

CONTINUED

ID: .....

*(2 marks)*

- e) The value of the string variable, `upi`, is always exactly four characters followed by exactly three digits, e.g., "afer023", "gbug006", "scam143". Complete the following section of code which prints the `upi` number (without any leading zeroes). For example, if the `upi` is "gbug006", the output is: `upi number: 6`

```
upi = ...      #A string of 4 letters followed by 3 digits
number = 0

print("upi number:", number)
```

*(2 marks)*

- f) Complete the following section of code which creates a tuple named `min_max_tuple` which contains 2 elements: the minimum number in the tuple, `a_tuple`, followed by the maximum number in the tuple, `a_tuple`. For example, if `a_tuple` is (3, 4, 15, 1), the output from the following section of code is:

```
(min, max): (1, 15).
```

```
#only some elements of the tuple are shown here
a_tuple = (5, 3, ..., 2, 1)

print("(min, max):", min_max_tuple)
```

*(2 marks)*

ID: .....

**Question 2 (15 marks)**

a) Given the following function:

```
def function_ifs(a, b, c):
    if a > b and c > b:
        print("A")
    elif a > b and c > a:
        print("B")
    else:
        if a % 2 == 0:
            print("C")
        elif b % 2 == 0:
            print("D")
        else:
            print("E")

    if not (a > b) or a > c:
        if a % 2 == 1:
            print("F")
        elif b % 2 == 1:
            print("G")
        print("H")
    elif not(a == c):
        print("I")

    print("J")
```

give the output of the following function call:

```
function_ifs(4, 6, 8)
```

*(3 marks)*

CONTINUED

ID: .....

b) Give the output when the following code is executed.

```
number = 10

while number > 2:
    number = number - 3

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

    print(number)
```

(3 marks)

c) Does the following code execute? If yes, give the output, otherwise write ONE sentence explaining why the code does not execute.

```
a_tuple = (3, 2, 8)

temporary = a_tuple[2]
a_tuple[2] = a_tuple[0]
a_tuple[0] = temporary

print("a_tuple:", a_tuple)
```

(3 marks)

CONTINUED

ID: .....

d) Given the following function:

```
def fiddle1(a_list):  
    list2 = a_list  
    list2[2] = list2[1] + list2[2]  
    a_list[0] = a_list[0] + a_list[1]
```

complete the output produced by the following section of code:

```
list1 = [4, 2, 1, 5]  
fiddle1(list1)  
print("list1:", list1)
```

```
list1: [          ]
```

(3 marks)

e) Given the following function:

```
def fiddle2(list1, list2):  
    list3 = list1  
    list3.append(list2[1])  
    list3.append(list2[0])  
    return list3
```

complete the output produced by the following section of code:

```
alist1 = [1, 3]  
alist2 = [2, 4, 6]  
alist2 = fiddle2(alist1, alist2)  
print("alist1:", alist1)  
print("alist2:", alist2)
```

```
alist1: [          ]  
alist2: [          ]
```

(3 marks)

CONTINUED

ID: .....

**Question 3 (16 marks)**a) Given the following `get_rent()` function:

```
def get_rent(is_couple, will_clean, is_good_cook):
    rent_basic_single = 180
    rent_basic_couple = 300
    will_clean_deduction = 40
    good_cook_deduction = 30

    rent = rent_basic_single
    if is_couple:
        rent = rent_basic_couple
    if will_clean:
        rent = rent - will_clean_deduction
    if is_good_cook:
        rent = rent - good_cook_deduction
    return round(rent)
```

complete the output produced by the following code.

```
rent = get_rent(False, False, True)
print("Rent: $" + str(rent))
```

Rent \$:

*(4 marks)*

ID: .....

- b) Complete the `get_random_word()` function which is passed one string parameter, `lots_of_words`. The words in the parameter string are separated from each other by a space character. The function returns a random word from the string of words. You can assume that the `random` module has been imported. For example, the following code:

```
word = get_random_word("car house holiday")
print(word)
```

prints either "car", "house", or "holiday".

```
def get_random_word(lots_of_words):
```

*(6 marks)*



ID: .....

- c) At a particular restaurant, each customer has a 75% chance of ordering a meat dish and a 25% chance of ordering a vegetarian dish. The `get_meat_vegie_numbers()` function is passed one parameter, the number of customers. The function carries out a simulation (for all customers) and returns a tuple containing the number of customers who order meat dishes followed by the number of customers who order a vegetarian dish. Complete the function. You can assume that the `random` module has been imported.

```
def get_meat_vegie_numbers(number_of_customers)
```

(6 marks)

CONTINUED

ID: .....

**Question 4 (13 marks)**

a) Complete the output produced when the following code is executed.

```
list1 = [9, 8, 2, 6, 3, 5]
print("list1 before:", list1)

list1 = list1[:3] + list1[4:]
print("list1 after:", list1)
```

```
list1 before: [9, 8, 2, 6, 3, 5]
list1 after: [          ]
```

(3 marks)

b) Complete the following code which removes the first 20 elements from `list1`. You can assume that `list1` contains at least 20 elements.

```
list1 = [ ... ] #The number of elements is greater than 20
```

```
list1 =
```

(3 marks)

c) Complete the function, `get_sum_list()` which is passed two list objects as parameters, `list1` and `list2`. The function returns a new list where each element is the sum of the first parameter list and the second parameter list in reverse order, i.e., the first element of the new list is the sum of the first element of `list1` and the last element of `list2`, the second element of the new list is the sum of the second element of `list1` and the second to last element of `list2`, and so on until the last element of the new list is the sum of the last element of `list1` and the first element of `list2`. You can assume that both parameter lists have the same length.

CONTINUED

ID: .....

For example, executing the following code with the completed function:

```
list1 = [3, 5, 3, 1, 4, 1]
list2 = [0, 3, 2, 4, 1, 6]
list3 = get_sum_list(list1, list2)

print("list1:", list1)
print("list2:", list2)
print("list3: ", list3)
```

gives the output:

```
list1 = [3, 5, 3, 1, 4, 1]
list2 = [0, 3, 2, 4, 1, 6]
list3: [9, 6, 7, 3, 7, 1]
```

```
def get_sum_list(list1, list2):
```

(7 marks)

CONTINUED

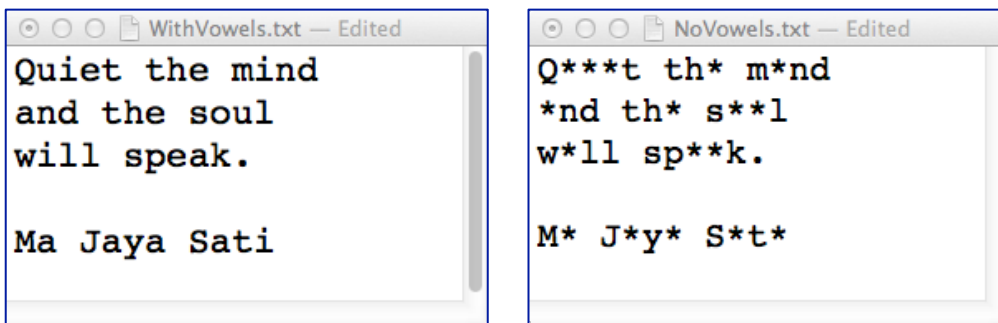
ID: .....

**Question 5 (15 marks)**

The following program reads the text from a file, "WithVowels.txt", replaces all the vowels in the text with "\*" symbols and writes the resulting text into a file named "NoVowels.txt". Complete the code in the three functions:

`read_from_file()`, `process_text()` and `write_to_file()`

so that the program executes correctly. A screenshot of the two files is shown below:



```
def main():
    #This function should not be changed
    text_to_process = read_from_file("WithVowels.txt")
    processed_text = process_text(text_to_process)
    write_to_file("NoVowels.txt", processed_text)

def read_from_file(filename):
    #Returns a string containing all the information read
    #from the file (the file name is given by the parameter).
```

CONTINUED

ID: .....

```
def process_text(text):  
    #Processes each character in the parameter string and  
    #returns a new string which is the same as the  
    #parameter string except that each vowel has been  
    #replaced by the "*" character.
```

```
vowels = "aeiouAEIOU"
```

```
def write_to_file(filename, information):  
    #Writes the parameter string, information, to the  
    #file (the file name is the first parameter).
```

```
main()
```

(15 marks)

CONTINUED

ID: .....

**Question 6 (14 marks)**

- a) Complete the `get_dict()` function which is passed two list parameters. The two parameters are both lists of strings and both have the same length. The function returns a dictionary object with the keys taken from the first list and the values taken from the corresponding element in the second list. For example, executing the following code with the completed function:

```
list1 = ["Romeo", "Tarzan", "Abelard", "Paolo"]
list2 = ["Juliet", "Jane", "Heloise", "Francesca"]
my_dict = get_dict(list1, list2)
print(my_dict)
```

gives the output:

```
{'Tarzan': 'Jane', 'Romeo': 'Juliet', 'Abelard': 'Heloise',
'Paolo': 'Francesca'}
```

```
def get_dict(key_list, value_list):
```

(7 marks)

CONTINUED

ID: .....

- b) Complete the `translate()` function which is passed two parameters. The first parameter is the dictionary which contains Italian words as the keys and the corresponding English words as the values. The second parameter is a string containing some Italian text. The `translate()` function breaks the Italian text into a list of words, gets the translation for each word from the dictionary and returns the corresponding English sentence. Note that if an Italian word is not in the dictionary, then the Italian word remains unchanged in the translation.

For example, executing the following code with the completed function:

```
#only part of the dictionary is shown here
a_dict = {"stanco":"tired", "gatto":"cat", "cane":"dog", ... }

message = "zitto il mio cane dorme e il mio gatto e stanco"
translation = translate(a_dict, message)
print("Message:", translation)
```

gives the output:

```
Message: quiet the mio dog sleeping e the mio cat e tired
```

```
def translate(dictionary, message):
```

(7 marks)

ID: .....

**Question 7 (9 marks)**

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

```
def draw_pattern(a_canvas):
    size = 10
    line_left = 0
    x_left = 0
    y_top = 0
    number_across = 5

    for row in range(0, 3):
        x_left = line_left
        is_square = True
        is_filled = True

        for col in range(0, number_across):
            rect_box = (x_left, y_top, x_left + size, y_top + size)
            if is_square:
                if is_filled:
                    a_canvas.create_rectangle(rect_box, fill="black")
                else:
                    a_canvas.create_rectangle(rect_box)
            else:
                if is_filled:
                    a_canvas.create_oval(rect_box, fill="black")
                else:
                    a_canvas.create_oval(rect_box)
            x_left += size
            is_square = not is_square
            is_filled = not is_filled

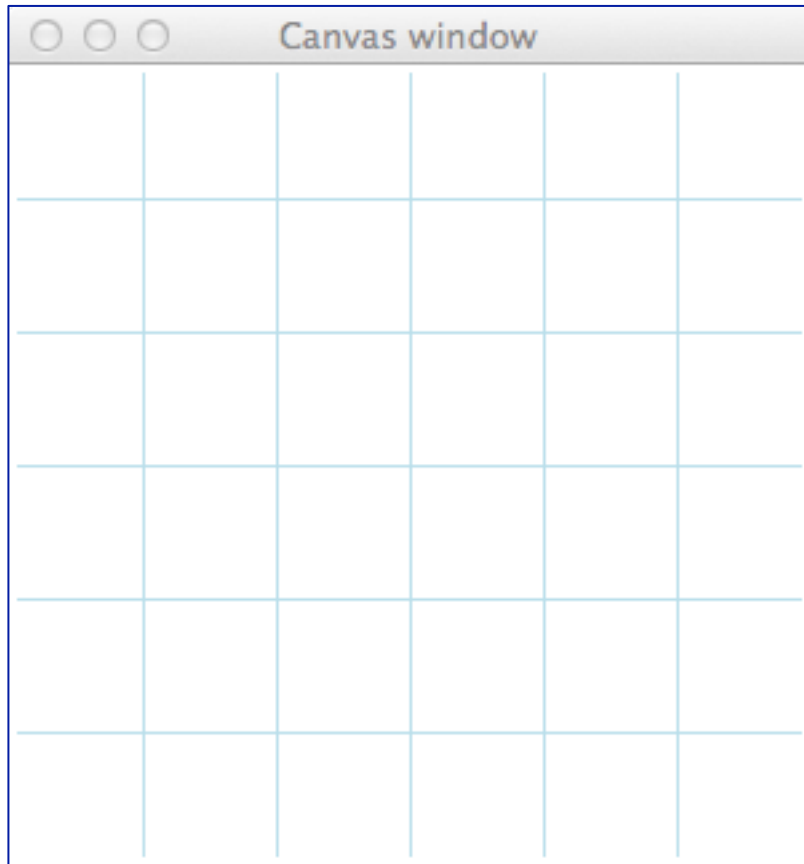
        y_top += size
        line_left += size
        number_across = number_across - 1
```

CONTINUED



ID: .....

Show the window:



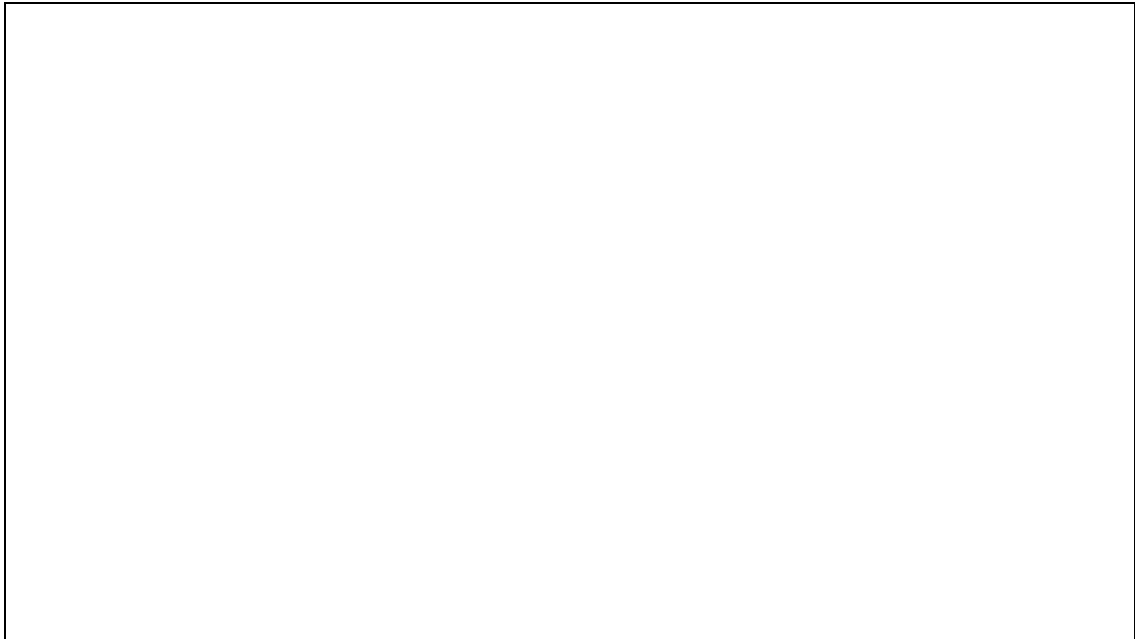
(9 marks)

ID: .....

**Question 8 (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 check_digit_count(number):  
    """Processes the digits in number
```

*(6 marks)*

```
    """
```

```
    sum = 0
```

```
    number_str = str(number)
```

```
    for single_digit in number_str:
```

```
        sum = sum + int(single_digit)
```

```
    return sum > 20
```

```
import doctest
```

```
doctest.testmod()
```

CONTINUED

ID: .....

**OVERFLOW PAGE**

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

ID: .....

**OVERFLOW PAGE**

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

ID: .....

**ROUGH WORKING (WILL NOT BE MARKED)**

(You may detach this page from the answer booklet and use it for rough working)

ID: .....

**ROUGH WORKING (WILL NOT BE MARKED)**

(You may detach this page from the answer booklet and use it for rough working)

