THE UNIVERSITY OF AUCKLAND

SECOND SEMESTER, 2014
Campus: City

COMPUTER SCIENCE

Principles of Programming

(Time Allowed: TWO hours)

NOTE:
You must answer **all** questions in this test.
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	
Forenames	
Student ID	
Login (UPI)	

Q1 (/14)	**Q4** (/11)	**Q7** (/12)	**Q10** (/5)
Q2 (/12)	**Q5** (/7)	**Q8** (/6)	**TOTAL**
Q3 (/9)	**Q6** (/14)	**Q9** (/10)	(/100)

ID: .....

Question 1 (14 marks)

a) Complete the output produced by the following code.

```
num1 = 7
num2 = 5
result = num2 + ((num2 * 2 - num1) ** 2) / 2
print("Result:", result)
```

```
Result:  9.5
```

(2 marks)

b) Complete the output produced by the following code.

```
num1 = 2
num2 = 7
result = str(num1) * (num2 // 2) + "0"
print("Result:", result)
```

```
Result: 2220
```

(2 marks)

c) Complete the following code which assigns a random number which is either 7, 8 or 9 to the variable, `number`. You can assume that the `random` module has been imported.

```
number =  random.randrange(7, 10)
```

(2 marks)

ID: .....

d) Given the following code:

```
num = 12
user_num = int(input("Enter the number: "))
num = num % user_num
print("Result:", abs(num - user_num) > 2)
```

complete the output produced when the user enters 5 at the prompt.

```
Enter the number: 5
Result: True
```

(2 marks)

e) Complete the output produced by the following code.

```
words = "beautiful day"
words = words[10:] + " " + words[:5]
print("Words:", words)
```

```
Words: day beaut
```

(2 marks)

f) Complete the output produced by the following code.

```
tuple1 = (5, 3, 1, 6, 7, 2, 4, 8)
tuple2 = tuple1[1: 6: 2]
print("tuple2:", tuple2)
```

```
tuple2: (3, 6, 2)
```

(2 marks)

CONTINUED

ID: .....

g) Complete the output produced by the following code.

```
tuple1 = (5, 3, 1, 6, 7, 2, 4, 8)
tuple2 = tuple1[5: 0: -2]
print("tuple2:", tuple2)
```

```
tuple2: (2, 6, 3)
```

(2 marks)

ID: .....

Question 2 (12 marks)

a) Complete the output produced by the following code.

```
a_list = [4, 7, 1, 9, 0, 6]
a_list[3] = a_list[3] - a_list[1]
a_list[1] = a_list[2] * a_list[5]
a_list[2] = a_list[a_list[4]]
print("List:", a_list)
```

List: [4, 6, 4, 2, 0 6]

(3 marks)

b) Given the following function definition:

```
def get_list(num1, num2):
    a_list = []
    for i in range(num2):
        number = num1 * i
        a_list.append(number)
    print(a_list)
```

show the values printed by the above function when the following call is made.

```
get_list(5, 4)
```

[0, 5, 10, 15]

(3 marks)

ID: .....

c) Complete the `starts_with_last_letter()` function which returns `True` if the parameter, `current_word`, starts with the same letter as the last letter of the parameter, `previous_word`. Otherwise, the function returns `False`. You can assume that both the parameter strings contain at least one letter.

```
def starts_with_last_letter(previous_word, current_word):
```

```
    last_previous = previous_word[-1]
    first_current = current_word[0]
    return last_previous == first_current
```

(3 marks)

d) The dictionary, `names_age`, is initialised as follows:

```
names_age = {"Giuseppe": 16, "Joe":5}
```

Write two assignment statements to add two new key:value pairs to the dictionary so that the dictionary contains the following four key:value pairs:

```
{'Giuseppe': 16, 'Joe': 5, 'Jill': 31, 'Ettie': 12}
```

```
names_age["Jill"] = 31
names_age["Ettie"] = 12
```

(3 marks)

ID: .....

Question 3 (9 marks)

a) Assume that the variable, `age`, has been assigned an integer value and that the variable, `is_full_time_student`, has been assigned a boolean value. Write **a boolean expression** which evaluates to `True` if `is_full_time_student` is `True` or if `age` is less than 19.

```
is_full_time_student or age < 19
```

(2 marks)

b) Assume that the variables, `x` and `y`, have been assigned integer values. Write **a boolean expression** which evaluates to `True` if `x` is non-negative and `y` is negative. Note that zero is a non-negative number.

```
x >= 0 and y < 0
```

(2 marks)

c) Assume that the variable, `goods_sold`, has been assigned an integer value. Write an **if statement** which assigns 10,000 to the variable, `bonus`, if the value of the variable, `goods_sold`, is greater than 500,000.

```
if goods_sold > 500000:
    bonus = 10000
```

(2 marks)

ID: .....

d) Assume that the variables, `age`, `minors`, `adults` and `seniors` have all been initialised to an integer value. Write an **if /elif statement** which:

adds 1 to the variable, `minors`, if the variable, `age`, is less than 18
adds 1 to the variable, `adults`, if the variable, `age`, is 18 through to 64 inclusive
adds 1 to the variable, `seniors`, if the variable, `age`, is 65 or older

```
if age < 18:
    minors += 1
elif age <= 64:
    adults += 1
else:
    seniors += 1
```

(3 marks)

ID: .....

Question 4 (11 marks)

a) Show the output when the following code is executed.

```
number = 6
while number > 0:
    number = number - 3
    print(number, end = ' ')
```

3 0

(3 marks)

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

```
y = 0
for i in range(0, 10, 2):
    y = y + i

print(y)
```

20

(3 marks)

ID: .....

c) Complete the `add_up()` function which computes and returns the following sum:

```
1/2 + 2/3 + 3/4 + ... + n/(n+1)
```

Note that the value of `n` is given as the parameter and that the function returns the result of the calculation.

```
def add_up(n):
```

```
top_num = 1
bottom_num = 2
sum = 0

while top_num <= n:
    sum += top_num / bottom_num
    top_num += 1
    bottom_num += 1

return sum
```

(5 marks)

ID: .....

Question 5 (7 marks)

Complete the `file_copy()` function which duplicates the content of a text file into a new file.

Note that the parameters, `source` and `target`, are the filenames of the original file and the new file respectively. The function returns the number of characters in the file.

```
def file_copy(source, target):
```

```
infile = open(source, 'r')

outfile = open(target, 'w')

contents = infile.read()
outfile.write(contents)

infile.close()
outfile.close()

return len(contents)
```

(7 marks)

ID: .....

Question 6 (14 marks)

a) Complete the `print_docket()` function which is passed two parameters:

a list of items,
and
a list of prices. The price of each item in the list of items is given by the corresponding price in the list of prices, i.e., the item in position 0 costs the price in position 0, the item in position 1 costs the price in position 1, etc.

This function goes through the list of items, printing the item and its price. As well, the function prints the total cost of all the items in the list. For example, executing the following code with the completed function:

```
items = ["sugar", "chocolate", "ice-cream", "bread"]
prices = [1.45, 2.30, 5.50, 3.25]
print_docket(items, prices)
```

produces the following output:

```
sugar 1.45
chocolate 2.3
ice-cream 5.5
bread 3.25
Total cost $12.5
```

```
def print_docket(item_list, price_list):
```

```
total = 0
i = 0
for item in item_list:
    print(item, price_list[i])
    total += price_list[i]
    i += 1

print("Total cost $ + str(total))
```

(7 marks)

b) Complete the `get_name_best_mark()` function which is passed one parameter:

a tuple which contains a string (a name), followed by one or more integers (marks out of 10), e.g., ("Jake", 6, 7, 9, 8)

The function returns a tuple made up of two values: the name, followed by the best mark. For example, executing the following code with the completed function:

```
name_best = get_name_best_mark(("Manu", 5, 7, 4, 8))
print(name_best)
print(get_name_best_mark(("Jake", 6, 7, 9, 8, 7, 2)))
print(get_name_best_mark(("Jane", 5, 2, 4)))
```

produces the following output:

```
('Manu', 8)
('Jake', 9)
('Jane', 5)
```

```
def get_name_best_mark(name_marks_tuple):
```

```
name = name_marks_tuple[0]
name_marks_tuple = name_marks_tuple[1:]

max_mark = max(name_marks_tuple)
return (name, max_mark)
```

(7 marks)

ID: .....

Question 7 (12 marks)

a) Complete the output produced by the following program:

```
def main():
    places_dict = {"A":"B", "C":"D", "B":"E", "D":"F", "E":"C"}
    place = "B"
    run = "B"
    is_present = True

    while is_present:
          if place in places_dict:
             place = places_dict[place]
             run = run + place
          else:
             is_present = False

    print("Run:", run)

main()
```

Run: **BECDF**

(5 marks)

b) Complete the `print_friendly_contacts()` function which is passed two parameters:

a dictionary of contacts. Each key:value pair in this dictionary is made up of a name and contact phone number, e.g., "Jim":4308866

and

a list of names.

The function prints each name from the list, followed by a " : ", followed by the contact number for this name. If the name is not present in the dictionary, then the name is printed followed by " : not found".

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

```
contacts_dict = {"Ann":5673254, "Jim":4308866,
                 "Li":8768192, "Ali":5679845, "Bob":4561978}
friends = ["Jim", "Ali", "Joe", "Bob"]
print_friendly_contacts(contacts_dict, friends)
```

produces the following output:

```
Jim : 4308866
Ali : 5679845
Joe : not found
Bob : 4561978
```

```
def print_friendly_contacts(contacts_dict, friend_list):
```

```
for friend in friend_list:
 if friend in contacts_dict:
   print(friend,":",contacts_dict[friend])
 else:
   print(friend,": not found")
```

(7 marks)

Question 8 (6 marks)

The definition of the `count_valid_chars()` function is given below:

```
def count_valid_chars(word):
    """
    >>> count_valid_chars("b")
    1
    >>> count_valid_chars("aaabbbcccabccab")
    15
    >>> count_valid_chars("aabbcftabaaca")
    10
    >>> count_valid_chars("aaa")
    3
    DONE!
    """

    result = 0

    for i in range(0, len(word)):
        if (word[i] == "a"):
            result = result + 1
        elif (word[i] == "b"):
            result = result + 1
        elif (word[i] == "c"):
            result = result + 1
        elif (word[i] == "_"):
            result = result + 1

    if (result <= 10):
        return result
    else:
        return 10
```

For each of the tests in the docstring, state whether it is successful and, if the test fails, state the reason.

```
>>> count_valid_chars("b")
1
```

successful

(1.5 marks)

ID: .....

```
>>> count_valid_chars("aaabbbcccabccab")
15
```

not successful – returns 10

(1.5 marks)

```
>>> count_valid_chars("aabbcftabaaca")
10
```

successful

(1.5 marks)

```
>>> count_valid_chars("aaa")
3
DONE!
```

not successful – 'Done!' is a comment and should be separated from the expected output by a blank line. Otherwise the expected output is

"3
Done!"

(1.5 marks)

ID: .....

Question 9 (10 marks)

a) Complete the following code which draws an **equilateral** triangle.

```
def print_equilateral_triangle(size):

    t = turtle.Turtle()
    t.forward(size)
```

```
t.left(120)
```

(2.5 marks)

```
    t.forward(size)
```

```
t.left(120)
```

(2.5 marks)

```
    t.forward(size)
```

b) Complete the `random_walk()` function which draws a random walk of n steps in 2 dimensions using a turtle. The step size is 20 pixels. The angle for each step is a random number of degrees between 0 and 359 inclusive. You can assume that the `random` module has been imported.

```
def random_walk(turtle, n):
```

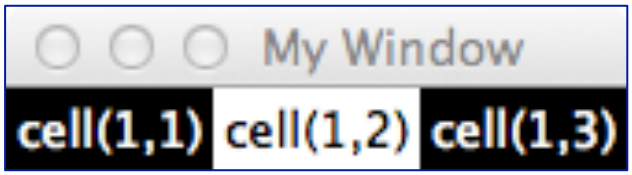
```
for step in range(0, n):

    angle = random.randrange(0, 360)
    turtle.setheading(angle)

    turtle.forward(20)
```

(5 marks)

ID: .....

Question 10 (5 marks)

a) Complete the `create_gui()` function which, using Tkinter, creates the GUI shown below. The window contains three labels alternately filled with black and white. The white label has a black foreground colour and the black labels have a white foreground colour.




```
def create_gui():
```

```
 window = Tk()

window.title("My Window")

label_11 = Label(window,text="cell(1,1)",
   background="black", foreground="white")
label_11.grid(row=0, column=0)

label_12 = Label(window,text="cell(1,2)")
label_12.grid(row=0, column=1)

label_13 = Label(window,text="cell(1,3)",
   background="black", foreground="white")
my_label_13.grid(row=0, column=2)
```

(5 marks)