THE UNIVERSITY OF AUCKLAND

FIRST SEMESTER, 2017
Campus: City

COMPUTER SCIENCE

Principles of Programming

(Time Allowed: TWO hours)

Note:
- The use of calculators is NOT permitted.

- You should separate the Section A Question Booklet from the Section B Question/Answer Booklet. You may keep the Section A Question Booklet. You must hand in the Section B Question/Answer booklet and the Teleform sheet.

- Compare the exam version number on the Teleform sheet supplied with the version number above. If they do not match, ask the supervisor for a new sheet.

- Enter your name and Student ID on the Teleform sheet. Your name and Student ID should be entered left aligned. If your name is longer than the number of boxes provided, truncate it.

- Answer **Section A** on the Teleform answer sheet provided. For Section A, use a dark pencil to mark your answers in the answer boxes on the Teleform sheet. Check that the question number on the sheet corresponds to the question number in this question/answer book. Do not cross out answers on the Teleform sheet if you change your mind. You must completely erase one answer before you choose another one. If you spoil your sheet, ask the supervisor for a replacement. There is one correct answer per question.

- Answer Section B in the space provided in the Section B Question/Answer Booklet.

- Attempt all questions. Write as clearly as possible. The space provided will generally be sufficient but is not necessarily an indication of the expected length. Extra space is provided at the end of the Question/Answer Booklet.

SECTION A

MULTIPLE CHOICE QUESTIONS

For each question, choose the **best** answer according to the information presented in lectures. Select your preferred answer on the Teleform answer sheet provided by shading in the appropriate box.

Question 1
[2.5 marks] What is the output produced by the following code?

```
value = (13 / 3 // 2) ** (2 % 6 ** 2) * 4 - 1 + 2
print(value)
```

    (a) `256.0`
    (b) `17.0`
    (c) `65.0`
    (d) `5.0`
    (e) None of the above.

Question 2
[2.5 marks] Given the code below:

```
import random
var1 = random.randrange(-2, 26 , 2)
var2 = random.randrange(51, -7, -6)
print(max(var1, var2))
```

which of the following values could be produced?
I `-4`
II `24`
III `17`
IV `0`

    (a) II and IV
    (b) I, II, III, and IV
    (c) I, II and III
    (d) I and III
    (e) II, III and IV

Question 3

[2.5 marks] What is the output produced by the following code if the user enters `"i am legend"` at the prompt?

```
sentence = input("Enter a sentence: ")
new_sentence = ""
while sentence.rfind(" ") != -1:
    index = sentence.rfind(" ")
    new_sentence += sentence[index+1:] + " "
    sentence = sentence[:index]
new_sentence += sentence
print(new_sentence)
```

    (a) `i am legendi am legend`

    (b) `legend am ii am legend`

    (c) `i am legend`

    (d) `legend am i`

    (e) No output will be produced as there is an error in the code

Question 4

[2.5 marks] What is the output produced by the following code?

```
def display_name(name):
    message = "Hello " + name + "."
    print(message, end = " ")

def display_date(date):
    message = "Today is " + date + "."
    print(message, end = " ")

def main():
    message = "Welcome!"
    print(display_name("Eddie"), end = " ")
    message = display_date("16 May 2017")
    print(message)

main()
```

    (a) `Hello Eddie. Welcome!`

    (b) `Hello Eddie. None Today is 16 May 2017. None`

    (c) `Hello Eddie. Today is 16 May 2017.`

    (d) `Hello Eddie. Today is 16 May 2017. Welcome!`

    (e) None of the above.

Question 5
[2.5 marks] What is the output produced by the following code?

```
def get_letter(score):
    if score >= 60:
        letter = "D"
    elif score >= 70:
        letter = "C"
    elif score >= 80:
        letter = "B"
    elif score >= 90:
        letter = "A"
    else:
        letter = "F"
    return letter

def main():
    print(get_letter(95), end = " ")
    print(get_letter(66), end = " ")
    print(get_letter(74), end = " ")
    print(get_letter(87), end = " ")
    print(get_letter(54), end = " ")

main()
```

(a) D D D D F
(b) A B C D F
(c) A D C B F
(d) D B D C F
(e) None of the above.

Question 6
[2.5 marks] What is the output produced by the following code?

```python
def fun_1(num):
    print("1.", num, end = " ")
    num += 20
    return (num - 10)

def fun_2(num):
    num = num - 15
    print("2.", num, end = " ")
    return (num + 10)

def main():
    num = 6
    num = fun_1(num)
    print("3.", num, end = " ")
    num = fun_2(num)
    print("4.", num, end = " ")

main()
```

(a) 3. 6 1. 26 4. 11 2. 9
(b) 1. 6 2. 1 3. 16 4. 15
(c) 1. 6 2. 16 3. 1 4. 20
(d) 1. 6 3. 16 2. 1 4. 11
(e) None of the above.

Question 7
[2.5 marks] What is the output produced by the following code?

```python
a_list = [4, 5, 6, 3, 4, 5]
for index in range(len(a_list) - 2, 1, -2):
    if a_list[index] - a_list[index - 1]  < 2:
        a_list.pop(index)
```

(a) [4, 5, 3, 5]
(b) [4, 5, 6, 3, 4, 5]
(c) [4, 5, 6, 4]
(d) [4, 6, 4]
(e) [5, 3, 5]

Question 8

[2.5 marks] Given the following code, which of the following correctly states the type of the three variables, `object1`, `object2` and `object3`?

```
a_tuple = (31, "5", 2.0)
a_list = [a_tuple[0], a_tuple[2], 'True']
a_dict = {'c': 14, 'b': 'False'}

object1 = a_tuple[2:3]
object2 = a_list[2] is a_tuple[1]
object3 = a_list[2] + a_dict['b']
```

(a) `object1: list,    object2: boolean, object3: boolean`

(b) `object1: tuple,   object2: boolean, object3: string`

(c) `object1: tuple,   object2: float,   object3: boolean`

(d) `object1: float,   object2: string,  object3: string`

(e) `object1: list,    object2: string,  object3: boolean`

Question 9

[2.5 marks] What is the output produced when the following `main()` function is executed?

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

def fiddle1(list1):
    list2 = list1
    list1.append(4)
    list2.append(5)
    list2 = list2[0: 2]
    list2.append(6)
```

(a) `BEFORE: [3, 2, 1]`
    `AFTER: [3, 2, 1, 4, 5]`

(b) `BEFORE: [3, 2, 1]`
    `AFTER: [3, 2, 1, 4]`

(c) `BEFORE: [3, 2, 1]`
    `AFTER: [3, 2, 1]`

(d) `BEFORE: [3, 2, 1]`
    `AFTER: [3, 2, 1, 4, 6]`

(e) `BEFORE: [3, 2, 1]`
    `AFTER: [3, 2, 1, 4, 5, 6]`

Question 10

[2.5 marks] What is the output produced when the following `main()` function is executed?

```python
def main():
    list1 = [3, 2, 1]
    tuple1 = (6, 4, 5)
    print("BEFORE:", list1, tuple1)
    fiddle2(list1, tuple1)
    print(" AFTER:", list1, tuple1)


def fiddle2(list1, a_tuple):
    list2 = list(a_tuple)
    list2[0] = 5
    list1[0] = 5
    list1 = list2
    list1[1] = list2[0] + 6
```

(a) `BEFORE: [3, 2, 1] (6, 4, 5)`
    `AFTER: [5, 11, 5] (6, 4, 5)`

(b) `BEFORE: [3, 2, 1] (6, 4, 5)`
    `AFTER: [5, 2, 1] (6, 4, 5)`

(c) `BEFORE: [3, 2, 1] (6, 4, 5)`
    `AFTER: [5, 11, 1] (6, 4, 5)`

(d) `BEFORE: [3, 2, 1] (6, 4, 5)`
    `AFTER: [3, 2, 1] (6, 4, 5)`

(e) `BEFORE: [3, 2, 1] (6, 4, 5)`
    `AFTER: [5, 2, 1] (5, 4, 5)`

Question 11
[2.5 marks] What is the output of the following code fragment?
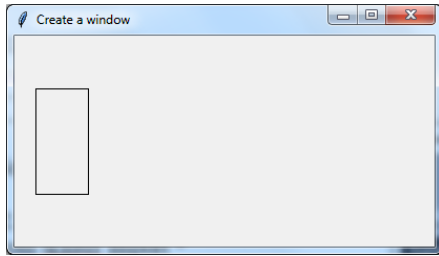
```
number = 4
for index in range(1, number + 1):
    print('.' * (number - index), end='')
    print(index)
```

(a) 1
    2
    3
    4

(b) ...1
    ...2
    ...3
    ...4

(c) ...1
    ..2
    .3
    4

(d) .1
    ..2
    ...3
   4

(e) None of the above.

Question 12

[2.5 marks] Consider the following screenshot and code fragment:


```
root = Tk()
root.title("Create a window")
root.geometry(_____) #Line A)
a_canvas = Canvas(root)
a_canvas.pack(fill=BOTH,expand=True)
a_rect = a_canvas.create_rectangle(_____) #Line B)
root.mainloop()
```

Which of the following statements could be used to replace **Line A)** and **Line B)** above to create a window of 400 pixels in width and 200 pixels in height and a rectangle at (20, 50) of size 50 pixels in width and 100 pixels in height?

(a) `root.geometry("200x400+50+100")`
    `a_rect = a_canvas.create_rectangle(20, 50, 70, 150)`

(b) `root.geometry(400, 200)`
    `a_rect = a_canvas.create_rectangle(20, 50, 70, 150)`

(c) `root.geometry("400x200")`
    `a_rect = a_canvas.create_rectangle(20, 50, 50, 100)`

(d) `root.geometry("400+200")`
    `a_rect = a_canvas.create_rectangle(20, 50, 50, 100)`

(e) `root.geometry("400x200+50+100")`
    `a_rect = a_canvas.create_rectangle(20, 50, 70, 150)`

Question 13

[2.5 marks] The `get_non_zero_average()` function below is not coded correctly. Four of the five doctests fail (i.e., the result returned by the function does not match the expected answer) and one of the five doctests passes. Which one of the doctest function calls below will pass, i.e., produces the expected answer?

```
def get_non_zero_average(list_of_numbers):
    """ the average of NON-ZERO numbers
    >>> get_non_zero_average([9, 2, 3, 8])
    5.5

    >>> get_non_zero_average([2, 0, 6, 2])
    3.3333333333333335

    >>> get_non_zero_average([9, 0, 2, 2, 8, 2, 2] )
    4.166666666666667

    >>> get_non_zero_average([0, 3, 9, 2, 2, 7, 0])
    4.6

    >>> get_non_zero_average([0, 3, 0, 5])
    4.0
    """

    return sum(list_of_numbers) / len(list_of_numbers)
```

(a) `>>> get_non_zero_average([9, 2, 3, 8])`

(b) `>>> get_non_zero_average([9, 0, 2, 2, 8, 2, 2] )`

(c) `>>> get_non_zero_average([0, 3, 9, 2, 2, 7, 0])`

(d) `>>> get_non_zero_average([0, 3, 0, 5])`

(e) `>>> get_non_zero_average([2, 0, 6, 2])`

Question 14

[2.5 marks] Consider the following function:

```python
def get_letter(words):
    my_dict = {}
    for ch in words:
        if ch != ' ':
            if ch in my_dict:
                my_dict[ch] += 1
            else:
                my_dict[ch] = 1

    my_list = []
    for letter, count in my_dict.items():
        my_list.append((count, letter))

    count, letter = max(my_list)
    return letter
```

What is the output of the following code fragment?

```python
print(get_letter('computer systems'))
```

    (a) `('s', 3)`
    (b) `s`
    (c) `3`
    (d) `[1, 2, 1, 3, 1, 2, 1, 1, 2, 1]`
    (e) None of the above.

THIS PAGE HAS BEEN INTENTIONALLY LEFT BLANK.

THE UNIVERSITY OF AUCKLAND

FIRST SEMESTER, 2017
Campus: City

Computer Science

Principles of Programming

(Time Allowed: TWO hours)

SECTION B Question/Answer Booklet

Answer all questions in this section in the space provided. If you run out of space then please use the Overflow Sheet and indicate in the allotted space that you have used the Overflow Sheet.

Surname:	
Forenames:	
Preferred Name **(if different to forenames)**	
Student ID:	
Login Name (UPI):	

MARKERS ONLY

Q1 – Q14	Q17	TOTAL
(/35)	(/13)	
Q15	Q18	
(/13)	(/13)	
Q16	Q19	
(/13)	(/13)	(/100)

Question 15 (13 marks)

a)   Complete the `censor_text()` function below which takes two string parameters, `text` and `word`. The function replaces all instances of `word` in `text` with a sequence of `"*"` characters of the same length as the length of `word`. You can assume that `text` and `word` will be strings containing only lowercase alphabetical characters, with a length of at least 1.

For example, when the following program is executed with the completed function, the output is:

```
**** this ****ed question
hello world
```

```python
def main():
  print(censor_text("darn this darned question", "darn"))
  print(censor_text("hello world", "drat"))

def censor_text(text, word):
```

```python
    stars = "*" * len(word)
    while text.find(word) != -1:
        index = text.find(word)
        text = text[:index] + stars + text[index + len(word):]
    return text
```

(6 marks)

```python
main()
```

b) The following program keeps prompting the user to guess a hidden number until the correct one is entered. At each incorrect guess the program will let the user know if the guess is too high or too low. Complete the following `guess_number()` function **using a `while` loop**, so that the program will execute correctly. A sample input/output is shown below.

```
Guess a magic number between 1 and 99.
Enter your guess: 55
Your guess is too low.
Enter your guess: 78
Your guess is too low.
Enter your guess: 95
Your guess is too high.
Enter your guess: 90
Yes, the number is 90.
```

```python
import random
def main():
    number = random.randrange(1,100)
    print("Guess a magic number between 1 and 99.")
    guess_number(number)

def guess_number(number):
```

```python
prompt = 'Enter your guess: '
feedback_low = 'Your guess is too low.'
feedback_high = 'Your guess is too high.'
feedback_exact = ' Yes, the number is'
guess = 0
while guess != number:
    guess = int(input(prompt))
    if guess < number:
        print(feedback_low)
    if guess > number:
        print(feedback_high)
print(feedback_exact, guess,".",sep="")
```
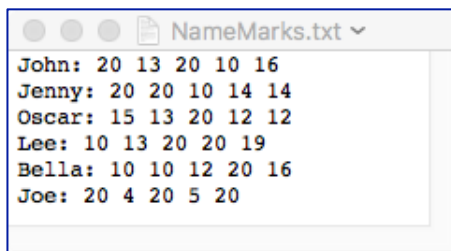
(7 marks)

```python
main()
```

Question 16 (13 marks)

The following program reads information from the input file, "NameMarks.txt", and writes the program results to the output file, "OscarMark.txt".

• The input file contains lines of names followed by several numbers with a colon space after each name (': '). Each line represents the name and assignment marks for one student.

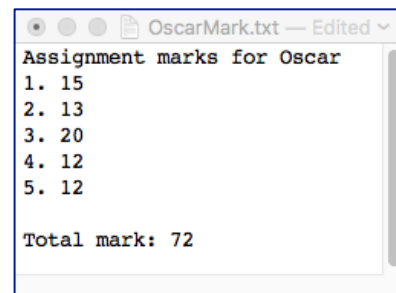• The output file displays the assignment marks and total for one student.

Below is an example of the "NameMarks.txt" file (on the left) and the "OscarMark.txt" file (on the right) produced by the completed program:

a) Complete the get_list_of_information() function which is passed one parameter, the name of a file which contains student names and marks with each student on a separate line. This function returns a list of strings. Each element of the list returned is a string containing the line of information for one student and contains no newline characters.

b) Complete the get_marks_tuple() function which is passed one parameter: a string containing marks (integers) separated by spaces. The function returns a tuple made up of each of the marks from the string. Each element of the tuple which is returned **must be an integer**. You can assume that the string passed as a parameter contains only whole numbers separated by blank spaces.

c) Complete the write_result() function which has three parameters: the name of the file, the student's name and a tuple containing integers. This function writes the following information (see the example output file above on the right) to the file:

• the string 'Assignment marks for " followed by the student's name,
• a numbered list of each of the marks from the parameter, marks_tuple,
• a blank line,
• the string 'Total mark: " followed by the total of all the marks.

```
def main():
    names_and_marks = get_list_of_information("NameMarks.txt")
    student_name = "Oscar"
    str_of_marks = get_student_marks_str(student_name, names_and_marks)
    tuple_of_marks = get_marks_tuple(str_of_marks)
    write_result("OscarMark.txt", student_name, tuple_of_marks)
```

```
def get_list_of_information(filename):

    input_stream = open(filename,"r")
    content = input_stream.read()
    content_list = content.split("\n")
    input_stream.close()
    return content_list
```

```
def get_marks_tuple(string_of_marks):

    list_of_marks = string_of_marks.split()
    mark_tuple = tuple()
    for mark in list_of_marks:
        mark_tuple += (int(mark),)
    return mark_tuple
```

```
def write_result(filename, student_name, marks_tuple):

    output_stream = open(filename,"w")
    output_stream.write("Assignment marks for " + student_name + "\n")
    count = 1
    total = 0
    for mark in marks_tuple:
        output_stream.write(str(count) + ". " + str(mark) + "\n")
        total += mark
        count += 1
    output_stream.write("\n" + "Total mark: " + str(total) + "\n")
    output_stream.close()
```

```
def get_student_marks_str(student_name, names_and_marks):
    for line in names_and_marks:
        if line.find(student_name) == 0:
            line_list = line.split(":")
            return line_list[1]
    return ""
```

```
main()
```

(13 marks)

Question 17 (13 marks)

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

```
def main():
    letters = '4-REAL'
    results = ''
    for character in letters:
        symbol = get_result_from_symbols_list(character)
        results = results + symbol

    print(letters, "-", results)


def get_result_from_symbols_list(letter):
    symbols_list = ['2ABC', '3DEF', '4GHI', '5JKL', '6MNO', '7PQRS',
                    '8TUV', '9WXYZ']
    for symbols in symbols_list:
        if letter in symbols:
            return symbols[0]
    return letter
```
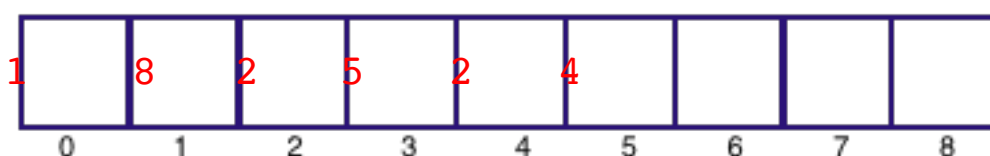
4-REAL – 4-7325

(3 marks)

b) 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 = [1, 2, 5, 6, 2]

a_list.insert(3, 2)
number = a_list.pop(-2)
number = a_list.index(2)
a_list.insert(number, 8)

a_list.append(a_list.pop() * 2)
```

1	8	2	5	2	4			
0	1	2	3	4	5	6	7	8

(3 marks)

c) Complete the `get_percentage_correct()` function which is passed two lists of strings as parameters. The two parameter lists have the same length and each element is a single uppercase letter. The first parameter is the list of correct answers and the second parameter is the list of user answers. The function returns the percentage of the user answers which are correct, rounded to the nearest whole number. For example, executing the following `main()` function with the completed `get_percentage_correct()` function, prints:

```
1. 75%
2. 60%
```

```
def main():
    correct_answers = ['A', 'E', 'B', 'A']
    user_answers =    ['A', 'E', 'B', 'E']
    result = get_percentage_correct(correct_answers,user_answers)
    print("1. ", result, "% ", sep='')


    correct_answers = ['A', 'E', 'B', 'A', 'C']
    user_answers =    ['A', 'E', 'D', 'E', 'C']
    result = get_percentage_correct(correct_answers,user_answers)
    print("2. ", result, "% ", sep='')


def get_percentage_correct(correct_answers, user_answers):
```

```
    num_q = len(correct_answers)
    correct = 0

    for i in range(len(correct_answers)):
        if correct_answers[i] == user_answers[i]:
            correct += 1

    percentage = round(correct/num_q * 100)
    return percentage
```

(7 marks)

Question 18 (13 marks)

a) Complete the `build_city_country_dict()` which takes a list of city:country strings as a parameter and returns a dictionary. Each element of the parameter list is a string made up of the city name, followed by a colon followed by the country name. The function processes the list, adds the information to a dictionary and returns the dictionary. The key of each dictionary item is the country and the value of each dictionary item is a list of cities in that country. For example, the following code:

```
country_list = ['London:United Kingdom', 'Chicago:United States',
  'Detroit:United States', 'Washington DC:United States',
  'Tokyo:Japan', 'Fukushima:Japan', 'Wellington:New Zealand',
  'Akaroa:New Zealand', 'Auckland:New Zealand',
  'Bay of Islands:New Zealand', 'Bay of Plenty:New Zealand']

my_dict = build_city_country_dict(country_list)
print(my_dict)
```

may print:

```
{'United States': ['Chicago', 'Detroit', 'Washington DC'], 'New
Zealand': ['Wellington', 'Akaroa', 'Auckland', 'Bay of Islands', 'Bay
of Plenty'], 'Japan': ['Tokyo', 'Fukushima'], 'United Kingdom':
['London']}
```

```
def build_city_country_dict(country_list):

    my_dict = {}
    for line in country_list:

        line = line.strip()
        line_list = line.split(":")
        country = line_list[1]
        city = line_list[0]
        if country in my_dict:
            if city not in my_dict[country]:
                my_dict[country].append(city)
        else:

            my_dict[country] = [city]

    return my_dict
```

(8 marks)

b) Complete the `display_num_cities()` function which takes a dictionary and a string as parameters. The function searches the dictionary for the country (passed as a parameter) and prints the number of cities which are in that country. If the country does not exist in the dictionary, the function prints `'NOT FOUND'`. For example, the following code fragment:

```
city_country_dict = {'United States': ['Chicago', 'Detroit',
'Washington DC'], 'New Zealand': ['Wellington', 'Akaroa', 'Auckland',
'Bay of Islands', 'Bay of Plenty'], 'Japan': ['Tokyo', 'Fukushima'],
'United Kingdom': ['London']}

display_num_cities(city_country_dict, 'New Zealand')
display_num_cities(city_country_dict, 'Canada')
display_num_cities(city_country_dict, 'United Kingdom')
```

prints:

```
New Zealand - 5
Canada - NOT FOUND
United Kingdom - 1
```

```
def display_num_cities(city_country_dict, country):
```

```
    if country in city_country_dict:
        list_of_cities = city_country_dict[country]
        print(country,"-",len(list_of_cities))
    else:
        print(country,"- NOT FOUND")
```

(5 marks)

Question 19 (13 marks)

Consider the following Python program:

```python
from tkinter import *


def main():
    window = Tk()
    window.geometry("600x600")
    a_canvas = Canvas(window)
    a_canvas.pack(fill=BOTH, expand=True)
    num_of_rows = 4
    left_hand_side = 10
    y_down = 10
    size = 10
    for row in range(num_of_rows):
        x_left = left_hand_side
        for col in range(row + 1):
            rect = (x_left, y_down, x_left + size, y_down + size)
            a_canvas.create_rectangle(rect) # Position A
            x_left += size
        points = [x_left, y_down, x_left + size, y_down,
                  x_left + (size // 2), y_down + size]
        a_canvas.create_polygon(points) # Position B
        y_down += size
    window.mainloop()


main()
```

a) In total, how many times is the statement marked **Position A** in the program above executed when the program is run?

row 0: executed ____1____ times

row 1: executed ____2____ times

row 2: executed ____3____ times

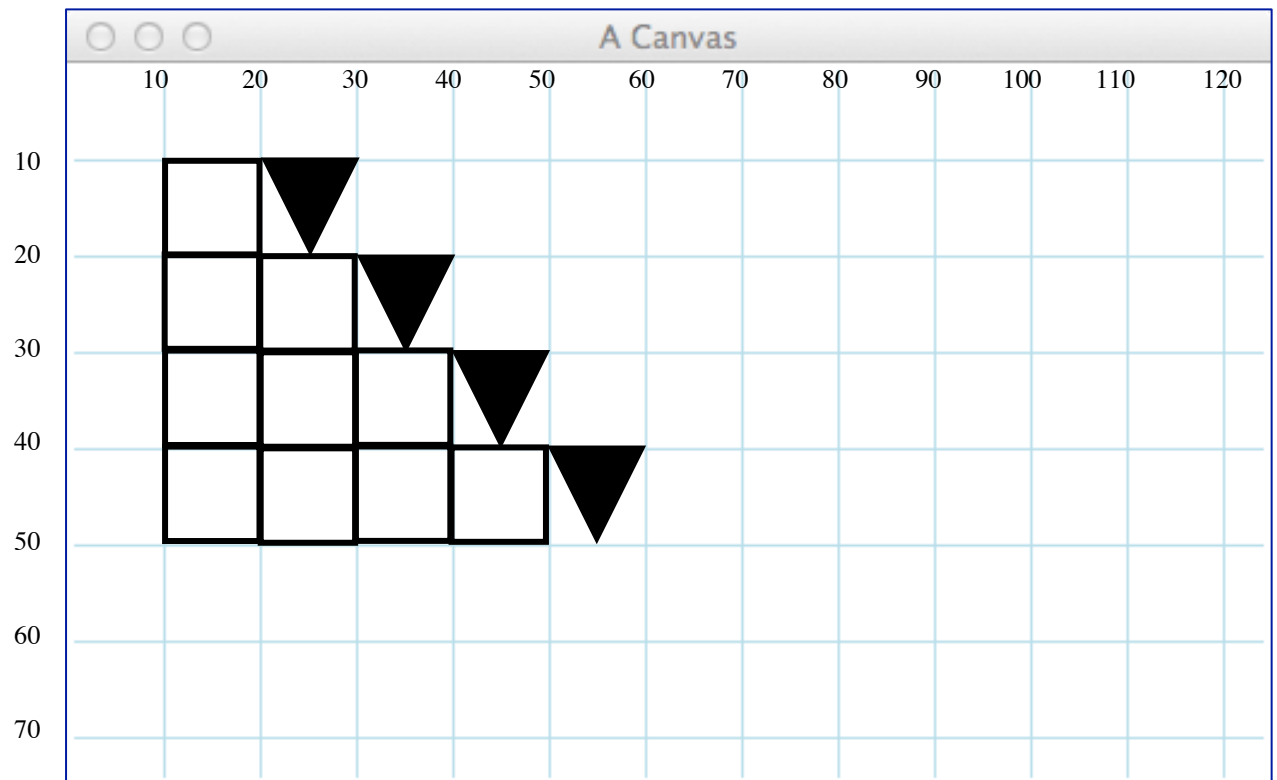row 3: executed ____4____ times

Total = _____10_____ times

(5 marks)

b) In total, how many times is the statement marked **Position B** in the program above executed when the program is run?

```
row 0: executed ____1____ times

row 1: executed ____1____ times

row 2: executed ____1____ times

row 3: executed ____1____ times

Total =_____4_____ times
```

(3 mark)

c) 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.

(5 marks)