

# THE UNIVERSITY OF AUCKLAND

---

FIRST SEMESTER, 2016

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 this exam book.

## 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] Which of the following could **NOT** be produced by the code below?

```
import random
var1 = random.randrange(19, 41, 4)
var2 = random.randrange(17, 45, 5)
var3 = random.randrange(42, 18, -3)
print(max(var1, var2, var3))
```

- (a) 21
- (b) 17**
- (c) 39
- (d) 32
- (e) 42

#### Question 2

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

```
value = 3 ** (1 + 1 * 2) // 5 % (20 / 2)
print(value)
```

- (a) 10.0
- (b) 6
- (c) 10
- (d) 5.0**
- (e) 5

**Question 3**

[2.5 marks] What is the output produced by the following code if the user enters “nothing is impossible” at the prompt?

```
string = input("Please enter a string: ")
left_space_index = string.find(" ")
right_space_index = string.rfind(" ")
if left_space_index != right_space_index:
    string = string[right_space_index + 1:] + " " + \
        string[left_space_index + 1:right_space_index] + \
        " " + string[:left_space_index]
print(string)
```

- (a) No output will be produced as there is an error in the code
- (b) is nothing impossible
- (c) impossible is nothing
- (d) nothing is impossible
- (e) None of the above.

**Question 4**

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

```
def function1(value1, value2):
    value3 = value1
    value1 = value2
    value2 = value3
    return value3

def main():
    value1 = 20
    value2 = 40
    print(function1(value1, value2), value2)

main()
```

- (a) 20 40
- (b) 40 20
- (c) 40 40
- (d) 20 20
- (e) None of the above.

**Question 5**

[2.5 marks] What is the output of the following code?

```
def function3(x, y, z):
    if x == 5 or y > 3:
        if x > 4 and z == 6:
            print("I")
        else:
            if y == 6 and z >= x:
                print("II")
            else:
                print("III")
    else:
        print("IV")

def main():
    function3(2, 4, 6)

main()
```

- (a) IV
- (b) II
- (c) III
- (d) I
- (e) None of the above.

**Question 6**

[2.5 marks] What is the output of the following code?

```
def function5():
    total = 0
    for number in range(1, 20):
        if number % 2 == 0 and number % 3 == 0:
            total += number
            print(number, end=" ")
    print(total)

def main():
    function5()

main()
```

- (a) 2 3 4 6 8 9 10 12 14 15 16 18 117
- (b) 5 10 15 30
- (c) 2 3 6 8 12 20 51
- (d) 6 12 18 36
- (e) None of the above.

**Question 7**

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

```
list1 = [3, 2, 4]

list1.append(7)
list1.append(1)

list1.insert(3, 5)

num1 = list1.pop(0)
num2 = list1.pop()

list1.insert(1, num1 + num2)

if 5 in list1:
    num3 = list1.index(5)
    list1.pop(num3)

print(list1)
```

- (a) [2, 4, 5, 4]
- (b) [2, 4, 4, 7]
- (c) [2, 4, 5, 4, 7, 1]
- (d) [2, 4, 5, 4, 7]
- (e) [2, 4, 4, 7, 1]

**Question 8**

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

```
list1 = [1, 2, 3, 4, 5, 6]

list2 = list1[:2] + list1[3::2] + list1[-5:-3:] + [list1[-2]]

print(list2)
```

- (a) [1, 2, 4, 6, 2, 3, 4, 5]
- (b) [1, 2, 4, 6, 2, 3, 4, 4]
- (c) [1, 2, 4, 5, 6, 2, 3, 5]
- (d) [1, 2, 4, 6, 2, 3, 5]
- (e) [1, 2, 4, 6, 3, 2, 5]

**Question 9**

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

```
def process_list(list1, number):  
    index = 0  
  
    while index < 4:  
        if list1[index] % number == 0:  
            list1.pop(index)  
            index = index + 1  
  
def main():  
    list1 = [1, 6, 3, 4, 9, 12]  
    process_list(list1, 3)  
    print(list1)  
  
main()
```

- (a) [1, 4, 12]
- (b) [1, 6, 3, 4, 9, 12]
- (c) [1, 3, 4, 12]
- (d) [1, 4]
- (e) [1, 4, 9, 12]

**Question 10**

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

```
def fiddle_lists(list1, list2):
    list1 = [1, 5]
    list2.append(list1[1])
    list1.append(list2[0])
    list1.append(list2[1])

def main():
    a_list1 = [5, 3, 2]
    a_list2 = [4, 6]

    fiddle_lists(a_list1, a_list2)

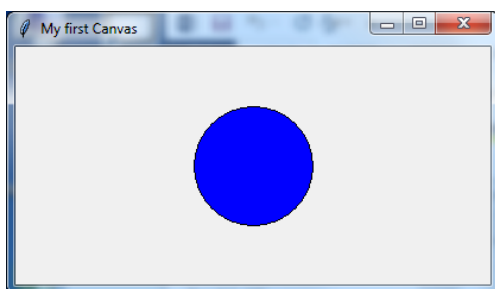
    print("a_list1:", a_list1)
    print("a_list2:", a_list2)

main()
```

- (a) a\_list1: [5, 3, 2, 4, 6]  
a\_list2: [4, 6, 5]
- (b) a\_list1: [5, 3, 2]  
a\_list2: [4, 6, 5]**
- (c) a\_list1: [5, 3, 2]  
a\_list2: [4, 6]
- (d) a\_list1: [1, 5, 4, 6]  
a\_list2: [4, 6]
- (e) a\_list1: [1, 5, 4, 6]  
a\_list2: [4, 6, 5]

**Question 11**

[2.5 marks] Consider a window that is 400 pixels wide and 200 pixels high, as shown below:



Which of the following method calls would correctly draw a circle that is positioned exactly in the centre of the window, and has a diameter of 100 pixels?

- (a) a\_canvas.create\_oval(50, 50, 150, 150, fill='blue')
- (b) a\_canvas.create\_oval(150, 50, 100, 100, fill='blue')
- (c) a\_canvas.create\_oval(50, 50, 100, 100, fill='blue')
- (d) a\_canvas.create\_oval(150, 50, 250, 150, fill='blue')**
- (e) None of the above.

**Question 12**

[2.5 marks] Consider the following function:

```
def q12(my_list, value):
    total = 0
    for i in range(len(my_list)):
        inner_list = my_list[i]
        for num in inner_list:
            if num > value:
                total += num
    return total
```

What is the output of the following code fragment?

```
my_list = [[15, 25, 80], [25, 120, 45], [70, 5, 130]]
print(q12(my_list, 100))
```

- (a) 250
- (b) 110
- (c) 400
- (d) 2
- (e) None of the above.

**Question 13**

[2.5 marks] Suppose the following code fragment:

```
def main():
    countries = {'usa': ("english", "Washington D.C."), \
                'poland': ("polish", "Warsaw")}

    #complete this
```

```
main()
```

produces

```
POLAND ('polish', 'Warsaw')
USA ('english', 'Washington D.C.')
```

Which of the following lines of code, if added to the `main()` function, would produce the above output?

- (a) 

```
for item in countries.items():
    print(item)
```
- (b) 

```
for k,v in countries:
    print(k.upper(), v)
```
- (c) 

```
for country in countries:
    print(country.upper())
```
- (d) 

```
for country, info in countries.items():
    print(country.upper(), info)
```
- (e) None of the above.



**Question 14**

[2.5 marks] What is the output of the following code fragment?

```
value = 0
for i in range(0, 25, 10):
    print(value, end=" ")
    for j in range(0, 10, 5):
        value += 1
print('final value', value)
```

- (a) 0 2 4 final value 6
- (b) 0 2 4 6 final value 8
- (c) 0 3 6 final value 9
- (d) 0 3 6 9 final value 12
- (e) None of the above.

**THIS PAGE HAS BEEN INTENTIONALLY LEFT BLANK.**

# THE UNIVERSITY OF AUCKLAND

---

**FIRST SEMESTER, 2016**

**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.

<b>Surname:</b>	
<b>First Name(s):</b>	
<b>Student ID:</b>	
<b>Login Name (UPI):</b>	

**MARKERS ONLY**

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

**Question 15 (13 marks)**

- a) Complete the `make_banner()` function below which takes a single string parameter that provides the information needed to print a banner. The string is formatted as follows:
- the last character of the string is used for the banner frame.
  - the second to last character of the string determines how wide the banner frame is on either side of the banner text. You can assume this will be a character between 0 and 9.
  - the remainder of the string is the banner text. You can assume that the banner text will be at least one character long.

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

```
#####
###COMPSCI 101 Exam###
#####
```

```
def main():
    make_banner("COMPSCI 101 Exam3#")
```

```
def make_banner(information):
```

```
    frame_char = information[-1]
    frame_width = int(information[-2])
    banner_text = information[:-2]
    print(frame_char * (len(banner_text) + 2 * \
                        frame_width))

    print(frame_char * frame_width + banner_text + \
          frame_char * frame_width)

    print(frame_char * (len(banner_text) + 2 * \
                        frame_width))
```

(6 marks)

```
main()
```

- b) Rewrite the `function3()` function shown below, using a `for ... in ... range()` loop. Both functions should behave in exactly the same way.

```
def function3(num):
    div = 1
    result = ""
    while div <= num:
        if num % div == 0:
            result = result + " " + str(div)
        div = div + 1
    return result
```

```
def function3(num):
```

(7 marks)

```
result = ""
for div in range(1,num+1):
    if num % div == 0:
        result = result + " " + str(div)

return result
```

**Question 16 (13 marks)**

a) Complete the `get_cheapest()` function below which takes two lists as parameters:

a list of strings, where each string is a description of an item,  
and,

a list of floats where the values represent the price of each item in the first parameter list.

The function returns a string containing information about the cheapest item in the list. The string returned by the function is made up of the item description followed by a '\$' sign and the price rounded to the nearest dollar. If two items have the same price, the cheapest is the **last** item from the beginning of the list. You can assume both lists have the same length and that they are not empty lists.

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

1. ['hat', 'shoes', 'bag', 'gloves', 'scarf']
2. [59.5, 275.75, 195.75, 65.15, 85.75]
3. hat \$60

```
def main():
    items = ["hat", "shoes", "bag", "gloves", "scarf"]
    prices = [59.5, 275.75, 195.75, 65.15, 85.75]

    info_cheapest = get_cheapest(items, prices)

    print("1.", items)
    print("2.", prices)
    print("3.", info_cheapest)
```

```
def get_cheapest(items, prices):
```

```
    index_of_cheapest = 0
    for i in range(1, len(prices)):
        if prices[i] <=
            prices[index_of_cheapest]:
            index_of_cheapest = i

    return items[index_of_cheapest] + " $" + \
           str(round(prices[index_of_cheapest]))
```

(7 marks)

```
main()
```

- b) Complete the `get_big_three()` function below which takes two tuples containing integer values as parameters and returns a tuple containing the three largest integer values from the combined values of the two parameter tuples. The numbers in the tuple returned by the function should be sorted from smallest to largest. You can assume that the two parameter tuples contain three or more elements when combined.

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

1. (2, 9, 8, 1)
2. (5, 11, 4)
3. (8, 9, 11)

```
def main():
    t1 = (2, 9, 8, 1)
    t2 = (5, 11, 4)

    t3 = get_big_three(t1, t2)

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

```
def get_big_three(t1, t2):
```

```
    t3 = t1 + t2
    t3 = sorted(t3)
    return tuple(t3[-3:])
```

(6 marks)

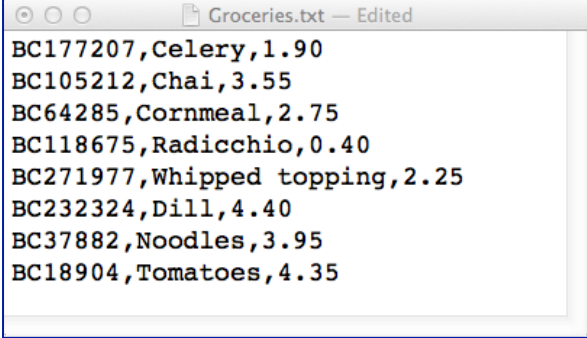
```
main()
```

### Question 17 (13 marks)

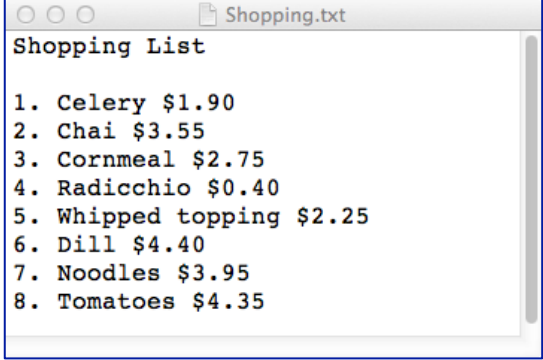
The following program reads information from the input file, 'Groceries.txt', and writes parts of the information to the output file, 'Shopping.txt'.

- Complete the `read_grocery_list()` function which takes a file name as a parameter and returns a list of lines read from the file. Note that none of the string elements of the returned list should contain a newline character ("`\n`").
- Complete the `write_numbered_list()` function which takes a file name and a list of strings as parameters. The first line written to the file is "Shopping List", followed by a blank line. Each element of the parameter list is a string containing three comma separated bits of information describing a grocery item, e.g., 'BC37882,Noodles,3.95', where the first part of the string is the item code, the second part is the item name and the last part is the item cost. For each element of the parameter list of strings, the function writes a number (starting from 1), followed by a full stop and a space, followed by the item name, followed by a space and the dollar sign, and lastly the cost of the item. Each string is written on a new line.

An example of a 'Groceries.txt' file (on the left) and the corresponding 'Shopping.txt' file (on the right) produced by the completed program are shown below:



```
BC177207,Celery,1.90
BC105212,Chai,3.55
BC64285,Cornmeal,2.75
BC118675,Radicchio,0.40
BC271977,Whipped topping,2.25
BC232324,Dill,4.40
BC37882,Noodles,3.95
BC18904,Tomatoes,4.35
```



```
Shopping List

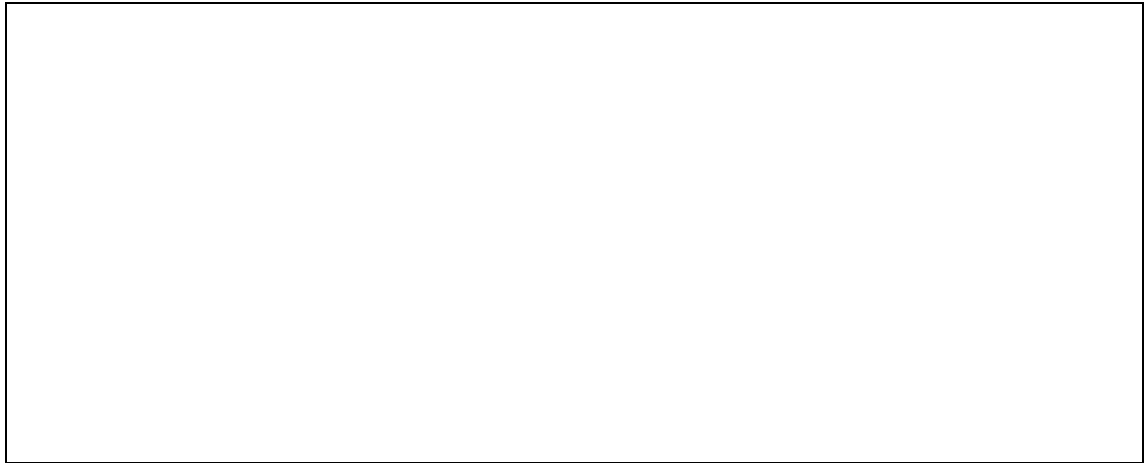
1. Celery $1.90
2. Chai $3.55
3. Cornmeal $2.75
4. Radicchio $0.40
5. Whipped topping $2.25
6. Dill $4.40
7. Noodles $3.95
8. Tomatoes $4.35
```

```
def main():
    grocery_list = read_grocery_list("Groceries.txt")
    write_numbered_list("Shopping.txt", grocery_list)
```

```
def read_grocery_list(filename):
```

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





(5 marks)

```
def write_numbered_list(filename, a_list):
```

```
    output_stream = open(filename, "w")
    output_stream.write("Shopping List\n\n")
    count = 1
    for item in a_list:
        details = item.split(",")
        output_stream.write(str(count) + ". " + \
details[1] + " $" + details[2] + "\n")
        count += 1
    output_stream.close()
```

(8 marks)

```
main()
```

**Question 18 (13 marks)**

a) Complete the `get_indexes()` function which takes two parameters:

`the_string`: a string of words separated by spaces (assume no punctuation or capitalization)  
`target`: a target word

The function returns a list containing the index positions of the target word in the string of words. For example, the following code:

```
print(get_indexes('all animals are equal but some animals are \
more equal than others', 'equal'))
```

prints

```
[3, 9]
```

because 'equal' appears at position 3, and position 9 in the string. (We start counting positions of words in the string from 0.)

```
def get_indexes(the_string, target):
```

```
    results = []
    str_list = the_string.split()
    for i in range(len(str_list)):
        if str_list[i] == target:
            results.append(i)
    return results
```

(6 marks)

- b) Complete the `build_index_dict()` function which takes a string as a parameter and returns a dictionary. The key of each dictionary item is the target word and the value of each dictionary item is the list of index positions of all occurrences of the target in the parameter string. For example, the following code:

```
my_dict = build_index_dict('all animals are equal but some \
animals are more equal than others')
print(my_dict)
```

may print:

```
{'than': [10], 'equal': [3, 9], 'animals': [1, 6],
'but': [4], 'all': [0], 'some': [5], 'are': [2, 7],
'more': [8], 'others': [11]}
```

Note: you **MUST** call the `get_indexes()` function defined in Part a) to solve this problem.

```
def build_index_dict(the_string):
```

```
    str_dict = {}
    str_list = the_string.split()

    for string in str_list:
        if string not in str_dict:
            str_dict[string] =
                get_indexes(the_string, string)

    return str_dict
```

(7 marks)

**Question 19 (13 marks)**

Parts a), b) and c) of this question refer to the following program:

```
from tkinter import *
def rectangular_grid(a_canvas):
    number_of_columns = 3
    number_of_rows = 5
    left_hand_side = 10
    y_down = 10
    size = 10
    dist = size * 3
    for i in range(number_of_rows):
        if i % 2 == 0:
            x_left = left_hand_side
        else:
            x_left = left_hand_side + size

        for j in range(number_of_columns):
            rect = (x_left, y_down, x_left + size, y_down + size)
            a_canvas.create_rectangle(rect, fill='blue')
            x_left += dist          #Position A
        y_down += size          #Position B

def main():
    root = Tk()
    root.title("A Canvas")
    root.geometry("400x300+10+20")
    a_canvas = Canvas(root)
    a_canvas.pack(fill = BOTH, expand = True)
    rectangular_grid(a_canvas)
    root.mainloop()

main()
```

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

15

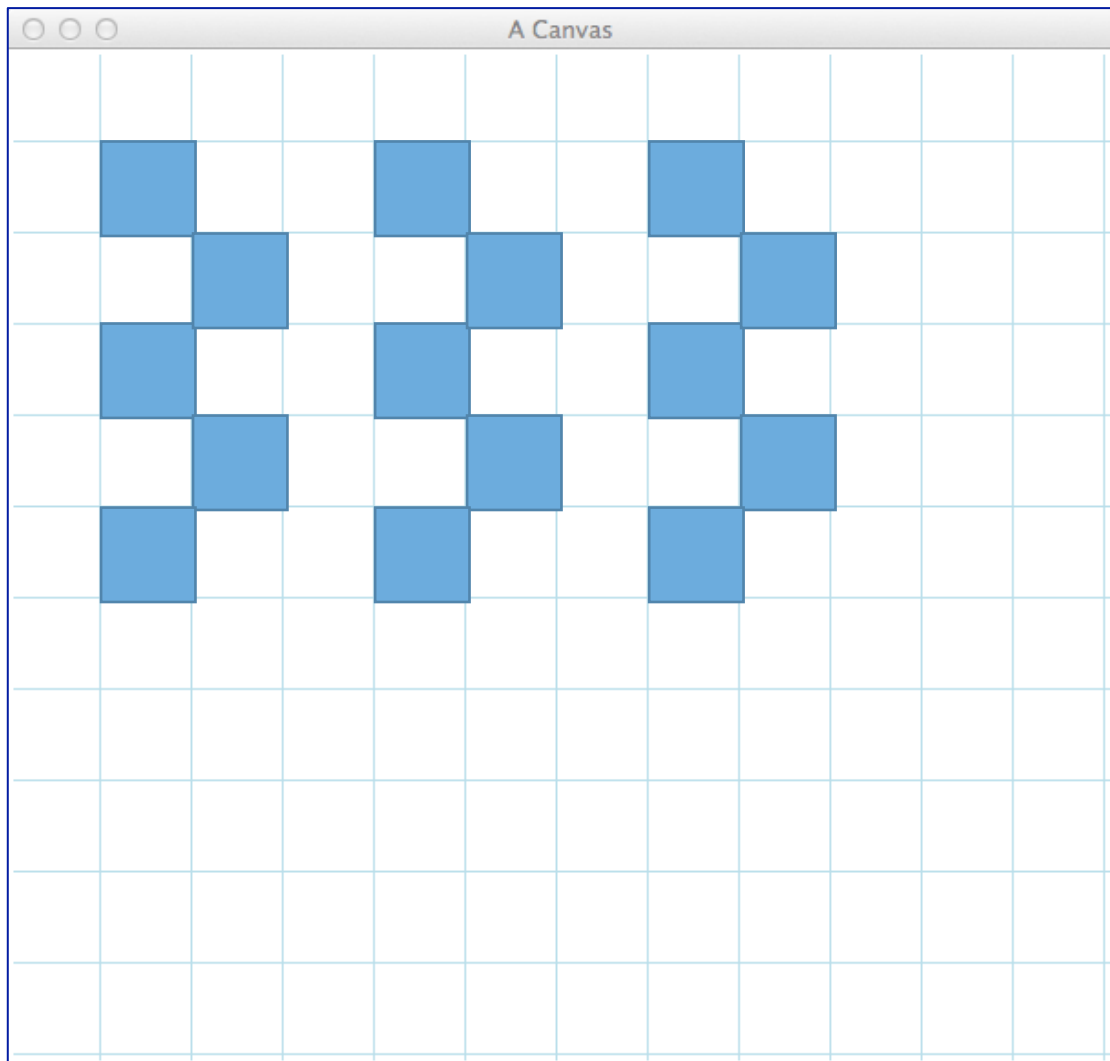
(2 marks)

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

5

(2 marks)

- 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.



(9 marks)