COMPSCI 101
Principles of Programming

Exam Revision

Documentation and Following Style Guidelines

In the docstring of the `get_result()` function below, add a short description (fifteen words or less) of the function.

```
def get_result(number, list_of_numbers):
    """
    Returns the value from the list_of_numbers
    parameter which is closest to the number
    parameter.
    """                                          (2 marks)
    result_number = list_of_numbers[0]
    smallest_difference = abs(result_number - number)
    for value in list_of_numbers:
        diff = abs(value - number)
        if diff < smallest_difference:
            smallest_difference = diff
            result_number = value

    return result_number
```

Documentation and Following Style Guidelines

- Rewrite the following function using descriptive variable and function names.

```
def a(b):
    a = 0
    for c in range(b + 1):
        a += c ** 2
    return a
```

```
def                    (              ):
```

```
def sum_of_squares(number):
    total = 0
    for i in range(number + 1):
        total += i ** 2
    return total
```

Output of Executing a Function and Code Tracing

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

```
def main():
    function_ifs(70, 45)

def function_ifs(num1, num2):
    if num1 < num2 and num2 < 60:
        print("A", end = " ")
        if num2 >= 10 and num2 % 2 == 0:
            print("B", end = " ")
        elif num1 % 10 < 3:
            print("C", end = " ")
        print("D", end = " ")
    else:
        if num1 > 50 or num2 < 50:
            print("E", end = " ")
        if num2 % 2 == 1:
            print("F", end = " ")
        print("G", end = " ")

    print("H")
```

```
E F G H
```

Output of Executing a Function and Code Tracing

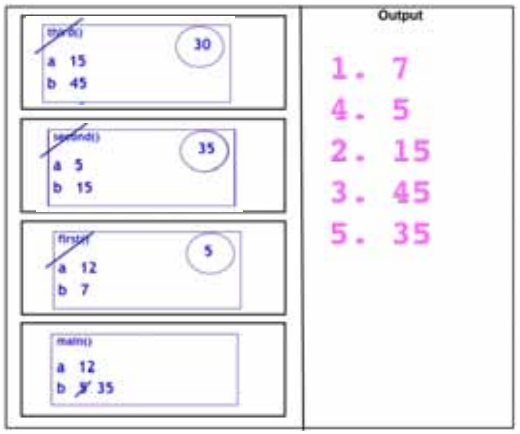
```
def first(a):
    b = a - 5
    print("1.", b)
    return a % b

def second(a):
    b = a + 10
    print("2.", b)
    return a + third(b)

def third(a):
    b = a * 3
    print("3.", b)
    return b - a

def main():
    a = 12
    b = first(a)
    print("4.", b)
    b = second(b)
    print("5.", b)

main()
```



Output of executing functions involving lists, tuples, and dictionaries, including passing mutable parameters

```
def main():
    a_list = [3, 4, 1]
    fiddle1(a_list)
    print("a_list:", a_list)

def fiddle1(list1):
    elements_to_add = [5, 5, 3]
    list2 = list1
    for element in elements_to_add:
        if element not in list1:
            list2.append(element)
    list1.pop(1)
```

a_list: [3, 1, 5]

```
def main():
    a_list = [3, 5, 7]
    fiddle2(a_list)
    print("a_list:", a_list)

def fiddle2(list1):
    list2 = list1
    list1 = [3, 4]
    list2.reverse()
```

a_list: [7, 5, 3]

Output of executing functions involving lists, tuples, and dictionaries, including passing mutable parameters

Given the following code, what is the type of each of the three Python objects object1, object2 and object3?

```
a_string = "MXQ339"
a_dict = {"A": "5", "M": [9, 3], "P": "M"}
a_list = [4, a_dict["P"], 2.5]

object1 = a_list.index(2.5)
object2 = a_dict[a_string[0]]
object3 = a_list[0] * a_list[-1]
```

object1 is of type: int

object2 is of type: list

object3 is of type: float

File output and the output of running a nested loop

```
def main():
    list1 = [4, 6, 7, 8, 1]
    the_list = [7, 6, 5, 4, 4, 7, 7, 2, 7, 6]
    count = process_lists(list1, the_list)
    print("count:", count, " the_list:", the_list)

def process_lists(list1, list2):
    count = 0
    for element in list1:
        while element in list2:
            index = list2.index(element)
            list2.pop(index)
        count = count + 1
    return count
```

count: 5 the_list: [5, 2]

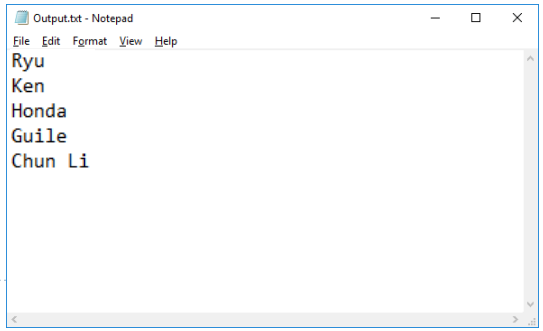
File output and the output of running a nested loop

What are the contents of the file "**Output.txt**" after the following program is run?

```
def main():
    data_tuple = ("Ken", "Ryu", "Guile", "Honda", "Chun Li")
    filename = "Output.txt"
    write_data(filename,data_tuple)

def write_data(filename,data_tuple):
    data_list = list(data_tuple)
    data_list.sort()
    data_list.reverse()
    output_stream = open(filename,"w")
    for item in data_list:
        output_stream.write(item + "\n")
    output_stream.close()

main()
```



Doctests

In the docstring of the `do_a_check()` function below, add ONE doctest which does not fail.

```
def do_a_check(value1, value2):
    """Checks the parameter values

    ---

    list_of_words = value1.split()
    return len(list_of_words) == value2

import doctest
doctest.testmod()
```

Boolean Expressions

Assume that the variables, `value1` and `value2` have both been assigned some integer value. Write a boolean expression which evaluates to `True` if `value1` is exactly divisible by `value2`. Otherwise the expression evaluates to `False`.

```
value1 % value2 == 0
```

Assume that the variable, `words`, has been initialised to some string. Write the boolean expression which tests if the variable, `words`, has at least five characters and ends with the letter "s".

```
len(words) >= 5 and words[-1] == "s"
```

Assume that the variables, `word1` and `word2` have both been assigned some string. Write a boolean expression which evaluates to `True` if both `word1` and `word2` contain the lowercase letter "a". Otherwise the expression evaluates to `False`.

```
"a" in word1 and "a" in word2
```

Assume that the variable, `value`, has been initialised to some integer. Write the boolean expression which tests if `value` is a two digit number and has a last digit (the right hand units digit) which is a 6.

```
len(str(value)) == 2 and value % 10 == 6
```

Tkinter

As accurately as possible, in the window below, show what is drawn when the `main()` function of the following program is executed. The grid lines have been drawn in the window to help you. The gap between adjacent gridlines is 10 pixels.

```
def draw_snake(a_canvas):
    left_hand_side = 20
    y_down = 30
    size = 10
    snake_list = [(20,30),(30,30),(40,30),(40,20),(40,10),(50,10)]
    number_of_elements = len(snake_list)
    for number_to_do in range(number_of_elements):
        x_left = snake_list[number_to_do][0]
        y_down = snake_list[number_to_do][1]
        rect = (x_left, y_down, x_left + size, y_down + size)
        a_canvas.create_rectangle(rect)
        a_canvas.create_oval(rect)

def main():
    ...
    draw_snake(a_canvas)
    root.mainloop()
```