CISC 101

HAND IN
Answers Are Recorded on Question Paper

QUEEN'S UNIVERSITY
SCHOOL OF COMPUTING

CISC101, FALL TERM, 2009
ELEMENTS OF COMPUTING SCIENCE I
FINAL EXAMINATION
15 December 2009

Instructor: Alan McLeod

Note for 2010 Students: Anything with a gray background covers a topic that is not applicable to the Fall 2010 Final exam.

This exam refers exclusively to the use of the Python language version 3. Comments are not required in the code you write. For full marks, code must be efficient as well as correct.

Please write your answers in the boxes provided. The back of any page can be used for rough work. This exam is 3 hours in length. Please put your student number at the top of each page. Extra space is provided on the second-to-last page of the exam.

An aid sheet has been appended to the exam. You may detach this page from the exam and do not have to return it when you hand in your exam, but the proctors can recycle the page for you.

This is a closed book exam. No computers or calculators are allowed or even needed.

PLEASE NOTE: "Proctors are unable to respond to queries about the interpretation of exam questions. Do your best to answer exam questions as written."

Student Number:

1.	/ 15	5.	/ 15
2.	/ 15	6.	/ 15
3.	/ 6	7.	/ 15
4.	/ 9		

TOTAL: / 90

Problem 1) [15 marks]
The following program runs without any errors. Write the output beside each print() statement:

```
def main():

    print(7 / 2)

    print(7 // 2)

    print(7.0 // 2.0)

    print(15 % 6)

    print(4 + 8 // 2)

    print((4 + 8) // 2)

    print(5 ** 2)

    print(5 > 2 and 6 == 5)

    print(7 != 7)

    print(10 <= 12 or 7 > 10 and 5 > 3)

    list1 = [1, 2, 3, 4, 5]
    list2 = [7, 8]
    print(list1 + list2)

    print(list2 * 3)

    print(list1[2])

    print(list1[1 : 4])

    print(list2[-1])
main()
```

Problem 2) [15 marks]
The following program runs without any errors. Write the output beside each non-empty print() statement:

```
def main():

    fodder = [1, 4.5, 6, 7, 10, 11]
    message = "Happy Holidays!"

    print(5 in fodder)


    print(5 not in fodder)

    for val in fodder:
        print(val, end=', ')

    print()     # prints a linefeed
    for val in reversed(fodder):
        print(val, end=', ')

    print()     # prints a linefeed
    del fodder[1]
    print(fodder)

    print(message.index('H'))


    print(message.count('H'))


    print(message.find('Z'))


    print('day' in message)


    print(message[-1].isalpha())


    print(message.lower())


    print(message)

    # There is a space in the quotes:
    print(message.partition(' '))

main()
```

Problem 3) [6 marks]

a) One advance in computer – assisted surgery is the ability to combine many different medical images to form a single 3D model to aid the surgeon in planning his/her operation. The surgeon can also use devices whose position can be tracked by sensors that locate passive markers (we saw a demo of such a device by Prof. Kunz). For the computer model to be used with the positional devices a third computer aided technique was developed. Name and describe this technique and the older process that it has replaced:

b) As Prof. Dingel pointed out, software is becoming increasingly complex with common operating systems being written with millions of lines of code. One technique that helps programmers like you deal with this complexity is the process of functional decomposition that we discussed in lecture. Name the three other techniques, or "weapons" listed by Prof. Dingel – his three "a" words. Which technique is used to shorten the "arrow of pain" between the requirements of a program and the expression of that program in machine-level code? Describe this technique.

Problem 4) [9 marks]
The following complete program sorts the list declared in the main function. It prints the list out in main and then at the end of each iteration of the outer loop in the sort, and then once again from main, after the list has been sorted:

```
def main():
    testList = [5, 7, 1, 2, 10, 3, 4, 6]
    print(testList)
    mysterySort(testList)
    print(testList)

def swap(numsList, pos1, pos2) :
    temp = numsList[pos1]
    numsList[pos1] = numsList[pos2]
    numsList[pos2] = temp

def mysterySort(numsList):
    i = 0
    size = len(numsList)
    while i < size - 1:
        aPos = i
        j = i + 1
        while j < size :
            if numsList[j] < numsList[aPos] :
                aPos = j
            j = j + 1
        if aPos != i:
            swap(numsList, i, aPos)
        i = i + 1
        print(numsList)
main()
```

What is the name of the sorting algorithm being used here?

What is the output of the program? The results of the first and last print statements are shown (the ones in main()); you must add the rest in the box to the right:

[5, 7, 1, 2, 10, 3, 4, 6]

[1, 2, 3, 4, 5, 6, 7, 10]

Problem 5) [15 marks]

Write two versions of a modified search function called “modSearch” that uses a sequential search to locate and return the start and end positions of all matches to a supplied target value in a supplied list. Your modSearch function header will be:

```
def modSearch(numsList, target)
```

You may assume that `numsList` will be in increasing order, will only contain numbers and will not be empty. You may also assume that `target` will contain a numeric value. Here is some code in a main() function, and its console output, that illustrates how your modSearch should work:

```
def main():
    testList = [2, 2, 2, 3, 4, 4, 4, 5, 5]
    print(modSearch(testList, 2))
    print(modSearch(testList, 3))
    print(modSearch(testList, 4))
    print(modSearch(testList, 5))
    try:
        print(modSearch(testList, 10))
    except ValueError as message:
        print(message)
```

Output:

```
(0, 2)
(3, 3)
(4, 6)
(7, 8)
Target not found.
```

As you can see, your function must raise a ValueError exception if the target value cannot be found in `numsList`.

Your first version of modSearch cannot use any list methods. The second version, which should be shorter, must use list methods. Write your code on the next page.

Problem 5, Cont.)

No list methods version of modSearch:

Version of modSearch using list methods:

Problem 6) [15 marks]

On the next page, write a function called loadData() that accepts a filename string as its only parameter. The function then opens the text file and loads the data from the file into a list of dictionaries, which will be returned by the function. The file consists of frog population data – the first column will be the name of the pond/marsh, the second column the type of frog and the third column an integer count. The three values are separated by a comma. Each dictionary in the list will represent a single row from the file. For example, if the file contains:

```
Lake Mead, green frog, 120
Swampy Muck, brown frog, 200
Warm Lake, hoppy frog, 1000
```

the function would return the list:

```
[{'count': 120, 'pondname': 'Lake Mead', 'frogname': ' green frog'},
{'count': 200, 'pondname': 'Swampy Muck', 'frogname': ' brown frog'},
{'count': 1000, 'pondname': 'Warm Lake', 'frogname': ' hoppy frog'}]
```

You may assume that the file will always be present, will not be empty and will always have the correct format, as shown above. However, the file can contain any number of rows. Do not write any methods other than loadData().

Problem 6, Cont.)

Problem 7) [15 marks]

Write a function called makeChange() that makes change for dollar amounts under $5.00, using only loonies, quarters, dimes and pennies. The function accepts the dollar amount and returns a dictionary consisting of the coins required to make this amount. Here are the coins to use:

loonie	$1.00
quarter	$0.25
dime	$0.10
penny	$0.01

If the dollar amount is less than or equal to zero or greater than or equal to 5.00 raise a ValueError exception. Otherwise your function must return a dictionary, where the keys are the coin names given above and the values are the numbers of each coin required. Don't include a coin if it is not required to make the amount (*ie*. the count for that coin would be zero).

You may need to use floor division: // and perhaps the modulo operator: % for this problem. Do not write any other functions. Here are a few examples of dollar amounts and the dictionary that would be returned for each:

```
1.00
{'loonies': 1}

3.50
{'loonies': 3, 'quarters': 2}

0.97
{'pennies': 2, 'dimes': 2, 'quarters': 3}

2.59
{'pennies': 9, 'loonies': 2, 'quarters': 2}
```

Hint: floor division and modulo do work as expected for float values, but consider removing the fractional amount from your input value by multiplying it by 100…

Write your function on the next page.

Problem 7, Cont.)

(*blank page*)

Some Built-In Functions:

abs(*number*)	# the absolute value of number
len(*obj*)	# the length of the iterable or string
str(*obj*)	# convert obj to a string
int(*number or string*)	# convert to an int
float(*number or string*)	# convert to a float
list(*obj*)	# convert to a list
set(*obj*)	# convert to a set where each element is unique
range(*[start,] stop [, step]*)	# creates an iterable used with a for loop
input(*stringPrompt*)	# returns a string from the console
print(*obj,…, sep=' ', end='\n'*)	# displays output to the console
chr(*unicode*)	# the character for the unicode value
ord(*character*)	# the Unicode value for the character
reversed(*obj*)	# a reversed iterable
sorted(*obj*)	# a sorted version of obj
isinstance(*obj, type*)	# True if obj is of the supplied type
max(*obj*)	# the highest value in the supplied iterable
min(*obj*)	# the lowest value
sum(*obj*)	# the sum of the numeric values in obj
open(*filename, mode*)	# open filename – mode is 'r', 'w' or 'a'

List Methods:

list.append(*obj*)	# appends obj to list
list.count(*obj*)	# counts occurrences of obj
list.index(*obj*)	# first occurrence of obj
list.index(*obj, i, j*)	# search between i and j
list.insert(*index, obj*)	# insert obj at index
list.pop()	# remove and return element at index = -1
list.remove(*obj*)	# search for, and remove obj
list.reverse()	# reverses in place
list.sort()	# sorts in place

File Object Methods:

fileobj.read()	# reads entire file
fileobj.readline()	# a single line
fileobj.readlines()	# a list of lines
fileobj.write(*str*)	# writes str to file
fileobj.close()	# close file object

Some String Methods:

# the number of occurrences of str
string.count(*str, beg=0, end=len(string)*)
# True if the string ends with str
string.endswith(*str, beg=0, end=len(string)*)
# replace tabs with spaces
string.expandtabs(*tabsize=8*)
# index location of str, -1 if not found
string.find(*str, beg=0, end=len(string)*)

string.format(*args*)	# args are placed and formatted into the string according to format codes

# index location of str, ValueError raised if not found
string.index(*str, beg=0, end=len(string)*)

string.isalnum()	# True if letter or numeric character
string.isalpha()	# True if letter
string.isdigit()	# True if numeric character
string.islower()	# True if lower case
string.isspace()	# True if whitespace (space, tab or linefeed)
string.istitle()	# True if titlecase
string.isupper()	# True if uppercase
string.join(seq)	# concatenate all strings in sequence
string.ljust(width)	# pad with spaces to width
string.lower()	# change all to lower case
string.lstrip()	# strip whitespace from start
string.partition(*str*)	# returns tuple of size 3 split around str

# replaces all occurrences of str1 with str2
string.replace(*str1, str2, num=string.count(str1)*)
# like find but searches from end
string.rfind(*str, beg=0, end=len(string)*)
# like index but searches from end
string.rindex(*str, beg=0, end=len(string)*)

string.rpartition(*str*)	# like partition but searches for str from end of string
string.rstrip()	# strips whitespace from end

# splits string into a list of pieces using str as delimiter
string.split(*str=" ", num=string.count(str)*)
# splits string into a list using linefeed as delimiter
string.splitlines(*num=string.count('\n')*)
# True if string starts with str
string.startswith(*str, beg=0, end=len(string)*)

string.strip()	# strip whitespace from beginning and end
string.swapcase()	# swaps letter case
string.title()	# titlecased version of string
string.upper()	# changes all to upper case