UNIVERSITY OF TORONTO
FACULTY OF APPLIED SCIENCE AND ENGINEERING

FINAL EXAMINATION, December 2010

DURATION: 2½ hours

CSC 180 H1F — Introduction to Computer Programming

Calculator Type: None
Exam Type: X

Examiner(s): F. Pitt & M. Guerzhoy

Student Number:	
Family Name(s):	
Given Name(s):	
Lecture Section:	☐ LEC 01 (with M. Guerzhoy) ☐ LEC 02 (with F. Pitt)

Do ***not*** *turn this page until you have received the signal to start.*
In the meantime, please read the instructions below *carefully*.

This final examination paper consists of 8 questions on 22 pages (including this one), printed on both sides of the paper. *When you receive the signal to start, please make sure that your copy is complete, fill in the identification section above, and write your student number where indicated at the bottom of every odd-numbered page (except page 1).*

Answer each question directly on this paper, in the space provided, and use the reverse side of the previous page for rough work. If you need more space for one of your solutions, use the reverse side of a page and *indicate clearly the part of your work that should be marked.*

Write up your solutions carefully! In particular, use notation and terminology correctly and explain what you are trying to do—part marks *will* be given for showing that you know the general structure of an answer, even if your solution is incomplete.

When writing code, assume all input is valid unless otherwise indicated (*i.e.*, do not waste time writing code to error-check). Comments and docstrings are not required except where indicated, although they may help us mark your answers. They may also be worth part marks if you cannot figure out how to write the code.

If you are unable to answer a question (or part), you will get 20% of the marks for that question (or part) if you write "I don't know" and nothing else—you will *not* get those marks if your answer is completely blank, or if it contains contradictory statements (such as "I don't know" followed or preceded by parts of a solution that have not been crossed off).

Marking Guide

# 1:	______/15
# 2:	______/ 8
# 3:	______/12
# 4:	______/10
# 5:	______/ 5
# 6:	______/ 7
# 7:	______/ 8
# 8:	______/12
TOTAL:	______/77

Good Luck!

Use this page for rough work—clearly indicate any section(s) to be marked.

CONT'D...

Question 1. [15 MARKS]

Part (a) Sequential Control Flow [1 MARK]

In the space on the right, show the output produced by running this code in the Python shell.

```
x = 2
print x
x = 3
print x
print x
```

Part (b) Variable Assignment [1 MARK]

In the code below, some of the first statement is not shown, for the purpose of this question. Assume the full statement is valid Python. Write a single Python statement where indicated, so that when the three statements are executed in the Python interpreter, `x` and `y` reference the same value.

```
x = ***(not shown but valid python)***
# Write your statement below this comment (and above the line y = z).




y = z  # y now references the same value as x
```

Part (c) Expressions [1 MARK]

In the space on the right, show the output produced by running this code fragment.

```
x = 2
print x + 1
print x
```

Part (d) `range` Function [1 MARK]

Write a single Python statement using the `range` function to print the list `[10,11,12,13,14]`.

Part (e) Accessing a List Element [1 MARK]

Write a single Python statement below the following code to print the value `"Toronto FC"`, by accessing the appropriate element from list `clubs`—*i.e.*, your answer must **not** just `print "Toronto FC"`.

```
clubs = ["Sounders", "Toronto FC", "Galaxy"]
```

Use this page for rough work—clearly indicate any section(s) to be marked.

CONT'D...

Question 1. (CONTINUED)

Part (f) Changing a List Element [1 MARK]

Write a single Python statement below the following code, so that it changes the second element of the list `items` from `"change me"` to `"changed value"`.

```
items = ["ignore", "change me", "ignore", "ignore"]
```

Part (g) Iterating over a List [1 MARK]

Assume you have a variable `courses` that refers to a list of strings. Write Python code to print each element of the list on a separate line.

Part (h) Nested Lists [1 MARK]

Suppose `L` refers to a list of lists. Using a `for` loop, print the second element of each of the sublists of `L`, each on its own line. For example, if `L` were set as follows:

```
L = [ ["-", "first", "-"], ["-", "second", "-", "-"], ["-", "third"] ]
```

your loop would print:

```
first
second
third
```

(Of course, your loop must work for any list of lists `L`, not just the one shown above.)

Part (i) Accumulating from a List [2 MARKS]

Assume you have a variable `temps` that refers to a non-empty list of temperatures (each one a floating point number). Write Python code to print the average of all the elements in `temps`, *without using the built-in* `sum` *function.*

Use this page for rough work—clearly indicate any section(s) to be marked.

CONT'D...

Question 1. (CONTINUED)

Part (j) Looping by Index [2 MARKS]

Assume you have a variable `L` that refers to a list of integers. Write Python code to **replace** each element of `L` with its square.

Part (k) Accessing a Dictionary Element [1 MARK]

Write a single Python statement below the following code to print the value associated with the key `"canada"` in the dictionary `currencies`.

```
currencies = {"us":"dollar", "canada":"dollar", "france":"euro"}
```

Part (l) Adding to a Dictionary [1 MARK]

Write a single Python statement that adds a new key-value pair to the dictionary `currencies` above. The new key should be `"brazil"` and it should be associated with the value `"real"`.

Part (m) Iterating over a Dictionary [1 MARK]

Assume you have a variable `d` that refers to a dictionary. Write Python code to print each **value** in the dictionary on a separate line—order does not matter.

Use this page for rough work—clearly indicate any section(s) to be marked.

CONT'D...

Question 2. [8 MARKS]

Each box in the left-hand column below contains a few lines of Python code. In the right-hand column, write down the value of variable `X` **after** the Python shell interprets the lines to the left—assume that the shell is fully reset before each block of code, *i.e.*, what happens in one block has no effect on what happens in the others. If interpreting the code would generate an error, circle the expression or statement that causes the error and write the reason that this expression or statement generates an error.

Code	Value of `X` (or "error" plus reason)
`s = "4"` `n = 7` `X = s + n`	
`X = ["two", "four"]` `y = X` `y.append("six")`	
`fav_col = {"F":"burgundy", "M":"sage"}` `X = fav_col["F"]` `fav_col["F"] = "lilac"`	
`fav_col = {"F":"burgundy", "M":"sage"}` `fav_col["F"] = fav_col["M"]` `fav_col["M"] = "navy"` `X = fav_col["F"]`	
`X = {"heads":[7,5], "tails":[2]}` `X["heads"] = X["tails"]` `X["tails"].append(9)`	
`y = range(5)` `X = y[1:4]` `y[2] = -1`	
`X = range(5)` `y = X[2:4]` `y[0] = -1` `X = X + y`	
`s = "abcde"` `t = s.substring(2, 4)` `t[0] = "Z"` `X = s + t`	

Use this page for rough work—clearly indicate any section(s) to be marked.

CONT'D...

Question 3. [12 marks]

The left-hand column in the table below shows a series of independent code fragments to be interpreted by the Python shell—similarly to the previous question, assume the shell is fully reset between any two blocks. For each code fragment, show the **expected output** in the right-hand column; if it would generate an error, circle the expression or statement that causes the error and write the reason that this expression or statement generates an error. Only put your final answer in the table.

Code	Output (or "error" plus reason)

`i = 2`
`price = [5.99, 1.99, 4.00, .20, 5.99]`
`if price[i] > price[0]:`
`   print "yes"`
`print "done"`	

`checked = True`
`i = 2`
`price = [5.99, 1.99, 4.50, .20, 5.99]`
`if checked and price[i] < 5.00:`
`   print "yes"`
`print "done"`	

`checked = True`
`x = 42`
`price = [5.99, 1.99, 4.50, .20, 5.99]`
`if checked or price[x] > 2.00:`
`   print "yes"`
`print "done"`	

`checked = True`
`i = 0`
`price = [1.99, 2.99, .50, 4.50]`
`while checked and i < len(price)-1:`
`   if price[i] < price[i + 1]:`
`      print price[i]`
`   else:`
`      print "**" + str(price[i])`
`      checked = False`
`   i += 1`
`print "done"`	

`dairy = ["milk","eggs","cheese"]`
`for item in dairy:`
`   item = item.upper()`
`   if len(item) >= 5:`
`      print item`
`print dairy`	

`IP = "   142.150.1.123  "`
`components = IP.strip().split(".")`
`print components`
`print int(components[0]) / 14`	

Use this page for rough work—clearly indicate any section(s) to be marked.

CONT'D...

Question 4. [10 MARKS]

Write a function `add_neighbours` that takes a list of integers and returns a new list, with the same number of elements as the original list but where each integer in the new list is the sum of its neighbours and itself (from the original list). For example, `add_neighbours([1,3,5,7])` returns `[4,9,15,12]` (which is equal to `[1+3, 1+3+5, 3+5+7, 5+7]`).

Use this page for rough work—clearly indicate any section(s) to be marked.

CONT'D...

Question 5. [5 MARKS]

Write a function `difference` that takes two dictionaries `d1, d2` and that returns a third dictionary `d` such that `d` contains every key that appears in both `d1` and `d2` but with different values (*i.e.*, where the values `d1[key]` and `d2[key]` are different). For each such `key`, you should set the value of `d[key]` to the tuple `(d1[key], d2[key])`. For example, if

```
    d1 = { "a": 1, "b": 2, "c": 3 }
    d2 = { "b": 2, "c": 4, "d": 6 }
```

then `difference(d1, d2)` returns `{ "c": (3,4) }`.

Use this page for rough work—clearly indicate any section(s) to be marked.

CONT'D...

Question 6. [7 MARKS]

The left-hand column in the table below contains different pieces of code that work with a list `L`. The details of what `L` contains are irrelevant to this question—assume that `L` has already been set. In the right-hand column, give a tight upper bound on the **complexity** of each piece of code, using big-Oh notation. Show your work.

Code	Complexity
`# L is a list with n = len(L)`	
`total = 0.0`	
`for item in L:`	
`   if item > 0.0:`	
`      total += item`	
`# L is a list with n = len(L)`	
`i = 0`	
`while i < n:`	
`    L[i] += i`	
`    i += 1`	
`while i > 0:`	
`    L[i] *= 2`	
`    i -= 2`	
`# L is a list with n = len(L)`	
`for i in xrange(n):`	
`   for j in xrange(i):`	
`      print L[j] - L[i]`	
`# L is a list with n = len(L)`	
`if L[0] % 2:`	
`   for i in xrange(n * n):`	
`      L[0] += i`	
`else:`	
`   for i in xrange(n):`	
`      L[0] -= i`	
`# L is a list with n = len(L)`	
`i = 1`	
`while i < n:`	
`   print L[i]`	
`   i *= 2`	
`# L is a list with n = len(L)`	
`for i in xrange(n):`	
`   if L[i] % 2:`	
`      for j in xrange(n):`	
`         L[j] += 1`	
`def f(L):`	
`   if len(L) == 0:`	
`      return 1`	
`   return L[0] * f(L[1:])`	
`# L is a list with n = len(L)`	
`f(L)`	

Use this page for rough work—clearly indicate any section(s) to be marked.

CONT'D...

Question 7. [8 marks]

Write a function `flatten` that takes a list `L` and that returns a list of all non-list elements nested within `L`, no matter how many sub-lists they are nested within. For example,

```
flatten([[]])                returns []
flatten([1])                 returns [1]
flatten([[2,3]])             returns [2,3]
flatten([[1,[]],[[[2]],3]]) returns [1,2,3]
```

Use this page for rough work—clearly indicate any section(s) to be marked.

CONT'D...

Question 8. [12 MARKS]

In HTML, all text that appears between the tags `<b>` and `</b>` is displayed in **bold**. Write a function `get_bold_text` that takes the URL of a web page (as a string) and that returns all of the bold text on that page (as a list of strings). For example, if the page at `http://url1.html` contains

```
<html><i>Hello</i> there!</html>
```

and the page at `http://url2.html` contains

```
<html>
This is a <b>simple</b> web page, with <B>NOTHING</b> complicated on it!
</html>
```

then

```
get_bold_text("http://url1.html") returns []
get_bold_text("http://url2.html") returns ["simple", "NOTHING"]
```

Remember that HTML tags are case-*insensitive*, *i.e.*, they can be uppercase (`<B>`, `</B>`) or lowercase (`<b>`, `</b>`) and the case of the closing tag does not have to match that of the opening tag—as in the second example above. Assume that the web page uses properly-formatted HTML (*i.e.*, all open tags are closed) and that there are **no** nested bold tags (*i.e.*, **nothing** like `"<b>bold <b>bolder?</b> bold</b>"`).

PLEASE WRITE NOTHING ON THIS PAGE