Name	Student Numbe

THE UNIVERSITY OF MANITOBA
COMP1012: Computer Programming for Scientists and Engineers
Midterm In Class Exam (A01—Andres)
2014 October 21, 8:30 am

Time: 45 minutes

Instructions:

1. Answer all questions on this paper. For multiple choice questions, circle the letter of the ***best*** or most complete choice. For short answer questions, write your answer in the space provided.
2. Extra work space is available on the last page.
3. You will find a Python Guide along with your midterm; ask if you don't have one. You may ***not*** use your own copy. No other aids (such as calculators or cell phones) are permitted.
4. You have 45 minutes to complete the exam.

Marks for Part 1	Part 2A	Part 2B	Part 3	Total
/ 4	/ 4	/ 4	/ 4	/16

Part 1: Predict the output [4 x 1 mark]

In each row of the table below, mentally execute the code on the left and enter the expected output in the box on the right. Each table row is separate. Use the space below for scrap work.

	Code Fragment	*Expected output*

A.	`What is printed by`
	`print 2 - 4**2 / 3 ?`	

B.	`What is printed by`
	`print tuple(range(-2))   ?`	

C.	`What is printed by`
	`print 4 or 5 != 5 / 2 ?`	

D.	`What is printed by`
	`print 1 < 2 >= 3 / 3 ?`	

Work space:

Name	Student Number

THE UNIVERSITY OF MANITOBA
COMP1012: Computer Programming for Scientists and Engineers
Midterm In Class Exam (A01—Andres)
2014 October 21, 8:30 am Time: 45 minutes

Part 2: Write a program [Total 8 marks]

2A. [4 marks] Write a complete program that generates and prints random lowercase letters between 'a' to 'z' until the last 4 characters printed form a valid word. The details are shown below, followed by two sample outputs, with different random characters:

- Include any imports needed by your program.
- Print the heading and final line of output as shown. Be sure to print the final period.
- Obtain a tuple of valid words by calling `getWords()`. Assume this function is already written; you do ***not*** have to write it.

```
GENERATE RANDOM STREAM

g j a i v y s a z e h r u q p i m m z l k h c y v y c h w j v v g n h i p s

The final word is hips.
```

```
GENERATE RANDOM STREAM

x g z z g y z d a i m u c n j k p m n y n s k x q n m n a j b k a t j q h z n n p j
i q g i i p s y k e y q s m n z s v f d o c k

The final word is dock.
```

For marker use only	
Item	Mark
A	
B	
C	
D	
E	
Sum	

Part 2: Write a program [Total 8 marks]

2B. [4 marks] The following series evaluates the function shown, for any x.

$$\frac{e^x - 1}{x} = 1 + \frac{x}{2!} + \frac{x^2}{3!} + \frac{x^3}{4!} + \frac{x^4}{5!} + \frac{x^5}{6!} + \ldots$$

Write a function `reducedExp` to evaluate the infinite sum above, using the standard approach taught in this course (NO FACTORIAL FUNCTIONS).

Details:

- `xx` is the value corresponding to x; `xx` could be positive or negative or 0.
- Assume `small` is a small positive number; add all the terms in the series that are greater than `small` in absolute value, and only those terms.
- Return the value of the series sum to the calling code.
- Do ***not*** print any output from this function.
- You do ***not*** need to fill in a doc string and you do ***not*** need to check parameter values.

```
def reducedExp(xx,small) :
```

For marker use only	
Item	Mark
A	
B	
C	
D	
E	
Sum	

THE UNIVERSITY OF MANITOBA	Name	Student Number

COMP1012: Computer Programming for Scientists and Engineers
Midterm In Class Exam (A01—Andres)
2014 October 21, 8:30 am Time: 45 minutes

Part 3: Circle the letter of the *best* answer, or provide the required answer [4 x 1 mark]

A. Given the following lines have just been executed, which of the options below creates a list of just the positive even numbers in seq1?

```
seq1 = [-3,  4, -4, -5,  4,  0,  1, -4,  3, -3]
seq2 = []
```

```
a)  for jj in seq1:  seq2.append(jj * (jj % 2 == 0) * (jj > 0))
b)  for jj in seq1:  seq2  = [jj] * (jj % 2 == 0) * (jj > 0)
c)  for jj in seq1:  seq2 += [jj] * (jj % 2 == 0) * (jj > 0)
d)  for jj in seq1:  seq2 += [jj] * ((jj % 2 == 0) or  (jj > 0))
e)  for jj in seq1:  seq2  = [jj] * ((jj % 2 == 0) and (jj > 0))
```

B. Given the following function definitions, which of the given choices produces the value 2 ?

```
def f(xx) : return xx - 1
def g(xx) : return xx * 2
def h(yy) : return yy // 2
```

```
a)  h(f(g(5)))
b)  f(h(g(5)))
c)  f(g(h(5)))
d)  g(f(h(5)))
e)  g(h(f(5)))
```

C. Which of the following statements about tuples, lists and strings is false, assuming `seq` is a tuple, a list or a string that is large enough to make the expressions valid?

a) `seq[3]` is the third item if `seq` is any these data types.
b) `seq[-2]` is the second last item if `seq` is any of these data types.
c) `bool(seq[3:3])` is `False` for any of these data types.
d) `2 * seq` is twice the size of seq for any of these data types.
e) `seq += seq` doubles the size of seq for any of these data types.

D. Using good coding practices and the same rules as QuizMaster, write a Python expression to evaluate this math expression, assuming math has already been imported:

$$\left\lfloor \left\lceil \left(10 + \tan\left(4 - \pi\right)\right)^{\frac{b}{3}} \right\rceil \right\rfloor$$

Put expression here

Python 2.7 Quick Reference Guide V1.2.11

(for more information see http://docs.python.org)

Enthought Canopy editor default layout

- Top window is a tabbed edit window—create a file, save it, and click the green triangle to run it
- Bottom window is a shell—enter commands for immediate execution, and see output of programs

Line length: max 80 characters

- **Long line:** have bracket open at line end
- **Error-prone alternative:** put `\` at end of line
- **Comment:** any text after unquoted `#` on a line

Data Types, Literals and Conversions

- **Integers**: optional sign with digits (no limit)
  `0, 314, -71322956627718, +6122233`
  `int(-5.7) → -5 # remove decimal part`
- **Floats**: decimal fraction, exponent (~16 digits)
  `0., 3.141592653589793, -7.132E8, 9.9e-20`
  `float(3) → 3.0 # convert to float`
- **Complex**: `z.real` and `z.imag` are always `floats`
  `0.0j, 3-7.132E8j, z = 2.1+0.9j`
  `complex(x,y) → x + yj # from 2 floats`
- **String**: single or double quotes allowed
  `'X', "I'd", '"No," he said.'`
  `repr(1 / 93.) → '0.010752688172043012'`
  `str(1 / 93.) →  '0.010752688172'`
- **Multi-line string**: triple quotes (single or double)
  `"""The literal starts here and goes on 'line after line' until three more quotes."""`
- **Boolean**: two aliases for 1 and 0 respectively:
  `bool(x)` → `True` or `False`
  Any zero or empty value can be used as `False` in a boolean expression; other values mean `True`
- `type("age") → <type 'str'>`

Math operators: `int` type result if `x` and `y` are `int`

- Power (*x*^*y*): `x**y`
- Times (*x*×*y*): `x * y`
- Divide by (*x*÷*y*): `x / y` or `x // y`
  `//` result *value* is always integer, but not *type*;
  `/` result value is integer when both `x` and `y` are.
- Remainder (*x* mod *y*): `x % y`
- Add (*x*+*y*): `x + y`
- Subtract (*x*–*y*): `x - y`

Operators with boolean result

- Compare: `<`, `<=`, `!=`, `==`, `>=`, `>`
- `x > y` → either `True` or `False` (1 or 0)
- `3 <= x < 5` means `3 <= x and x < 5`
- `x is y` means `x, y` refer to the same object (T/F)
- `x in y` means `x` is found inside `y` (T/F)

Evaluation Order from High Priority to Low

- `**`: `-2**2**3` is like `-(2**(2**3))`
- `+,-`: `-++-+3` is like `-(+(+(-(+3))))`
- `*,/,//,%`: `8 / 3 // 2 * 3 % 2` is like
  `(((8 / 3) // 2) * 3) % 2`
- `+,-`: `8 / 3 * 4 + -2**4 % 5` is like
  `((8 / 3) * 4) + ((-(2**4)) % 5)`
- `<,<=,!=,==,>=,>,is,in`:
  `5 + 2 < 3 * 8` is like `(5 + 2) < (3 * 8)`

Identifiers

- **Variables** (mixed case) can change value
  `sumOfSquares = 0.0`
- **Constants** (all uppercase with _) should stay fixed:
  `SECS_PER_MIN = 60`

Assignment

- `x = y` makes identifier `x` refer to the same object that `y` refers to
- `x,y = a,b` is like `x = a; y = b`
- `x,y = y,x` swaps values of `x` and `y`
- `x += a` is like `x = x + a`
- `x -= a` is like `x = x - a`
- similarly for `*=, /=, //=, %=, **=`

Output with old style formatting

- Command: `print 3,5,(1,2)` displays blanks between
  `3 5 (1, 2)`
- `"%d and %f: %s" % (3,5.6,"pi")`
  → `"3 and 5.600000: pi"`
- Conversion specifiers:

`%s`	string version, same as `str(x)`
`%r`	best representation, same as `repr(x)`
`%c`	shows number as a character, same as `chr(x)`
`%d`	an integer
`%0nd`	an integer left padded with zeros, width *n*
`%f`	decimal notation with 6 decimals
`%e, %E`	scientific notation with **e** or **E**
`%g, %G`	compact number notation with **e** or **E**
`%nz`	format **z** right-adjusted, width *n*
`%-nz`	format **z** left-adjusted, width *n*
`%n.mz`	format **z**: *m* decimals, width *n*
`%%`	a literal **%** sign

Input from user

- `var = raw_input("Enter value: ")`
- `intVal = int(var) # integer cast, or`
- `anyVal = eval(var) # could be dangerous, or`
- `assert '@' in var, "%s not e-mail address" % var`

Operating system functions:

```
import os, sys # operating system, system
os.chdir(sys.path[0]) # change to script file folder
print os.listdir('.') # file list of current folder
```

Built-in functions: `dir(__builtins__)`

- `abs(x) → |x|`
- `chr(35) → '#'`
- `dir(x)` → attributes of `x`
- `help(x)` → help on using `x`
- `len(x)` → length of `x`
- `max(2.1, 4, 3) → 4    # largest argument`
- `min(2.1, 4, 3) → 2.1  # smallest argument`
- `ord('#') → 35  # ASCII order`
- `range(a,b,c)` *see lists*
- `round(3.6) → 4.0 # nearest integer`
- `round(3.276,2) → 3.28 # 2 decimals`
- `sum( (1, 5.5, -8) ) → -1.5 # add items`
- `zip(listx, listy)` → list of (*x*,*y*) tuples

Math functions: `dir(math)`

`import math as m  # for float, or`
`import cmath as m # for complex`
To avoid "`m.`": `from math import *`

- `m.acos(x)` → inverse cosine [radians]
- `m.asin(x), m.atan(x) # similar`
- `m.ceil(x)` → least integer ≥ `x` [`math` only]
- `m.cos(x)` → cosine of `x` given in radians
- `m.e` → 2.718281828459045
- `m.exp(x)` → e^x
- `m.factorial(x)` → `x!` [`math` only]
- `m.floor(x)` → biggest int ≤ `x` [`math` only]
- `m.log(x)` → natural log of `x`
- `m.log10(x)` → log base 10 of `x`
- `m.pi → 3.141592653589793`
- `m.sin(x), m.tan(x) # see m.cos`
- `m.sqrt(x)` → square root of `x`

`import random # for pseudorandom numbers`

- `random.random()` → uniform in [0,1)
- `random.seed(n) # resets random number stream`
- `random.randrange(a,b)` → uniform int in [a,b)

`while` control structure

- `while` statement followed by indented lines
- `while` *condition* :
  —loops while *condition* is `True` (i.e., not 0)
  —indented lines are repeated each iteration
  —to terminate, make *condition* 0/`False`/empty
-
```
leftToDo, total = 100, 0.0
while leftToDo :  # i.e., while leftToDo > 0:
    total    += 1.0 / count
    leftToDo -= 1
```

Lists and tuples

- Both are sequences of arbitrary objects with index positions starting at 0, going up by 1
- A tuple has optional round brackets
- `xt = 1,    # a single member tuple`
- `xt = (1, 8.9, "TV")`
- A list has square brackets (required)
- `xl = [1, 8.9, "TV"]`
- `list("too") → ['t','o','o']`
- `tuple(["age",5]) → ('age',5)`

What you can do with a tuple

- You cannot change length or contents (***immutable***)
- `len(xt) → 3`
- `8.9 in xt → True`
- `xt.count(8.9) → 1 # how many 8.9 entries`
- `xt.index("TV") → 2 # first TV in position 2`
- `xt[2] → 'TV'    # in position 2`
- `xt[-1] → 'TV'   # in last position`
- `xt[0:2] → (1, 8.9)    # extract slice`
- `xt[1:] → (8.9, 'TV')  # open-ended slice`
- `xt[0: :2] → (1, 'TV') # slice by twos`

What you can do with a list

- almost anything you can do with a tuple, plus...
- `xl.append(5.7)   # adds 5.7 to end`
- `xl.insert(3,"x") # put "x" before xl[3]`
- `xl.pop(1) → 8.9 # remove 2nd entry`
  → `xl` is now `[1, 'TV', 'x', 5.7]`
- `xl.reverse() # reverses order`
- `xl.sort()    # in increasing order`
  → `xl` is now `[1, 5.7, 'TV', 'x']`
- `xl[1] += 2.2   # updates entry value`
- `xl[:2] = [2,3] # assign to slice`
- `xl[:] → [2, 3, 'TV', 'x'] # a copy`
- `range(5) → [0, 1, 2, 3, 4]`
- `range(2,5) → [2, 3, 4]`
- `range(2,15,3) → [2, 5, 8, 11, 14]`

Third element of a slice is a step size:

- `range(50)[::9] → [0, 9, 18, 27, 36, 45]`

`for` control structure

- `for` statement followed by indented lines
- `for item in listOrTupleOrString :`
  —`item` takes on the value of each entry in turn
  —indented lines are repeated for each `item`

-
  ```
  total = 0.0
  for number in range(1,101):
      total += 1.0 / number
  ```
- Parallel lists `list1` and `list2`:
  ```
  for e1, e2 in zip(list1, list2):
      print e1, e2
  ```
- If you need both position and value of `list` entry:
  ```
  for pos, value in enumerate(list1):
      print "Entry at %d is %g" % (pos,value)
  ```

List comprehension (embedded `for`)

- To generate one list from items in another
- `list2 = [f(n) for n in list1]`
- `squares = [n * n for n in range(90)]`

Defining functions

- `def` statement followed by indented lines
- `def myFunc(parm1, parm2, …) :`
  —creates a function `myFunc` with parameters
  —*`parmj`* is given the value used in calling `myFunc`
  —indented lines are executed on call
  —`return y` returns the value `y` as the results of the call

-
  ```
  def vectorLength(x,y):
      return m.sqrt(x * x + y * y)
  ```
- `vectorLength(3,4) → 5.0`

-
  ```
  def first3Powers(n): # multiple returns
      return (n, n * n, n * n * n)
  ```
- `x1, x2, x3 = first3Powers(6.2)`

Local and global variables

- a variable defined in the Python shell is *global*
- you can access it and *use* its value in a function
- a variable *<u>assigned</u> a value* in a function definition is *local*; it exists only during function execution
- *declare* a variable global inside a function and then you can assign to the global variable of that name:
  `global CHARS; CHARS = list("abc")`

`if - else` blocks

- `if` *condition* `:    # if statement`
  indented lines
- `elif` *condition* `:  # elif optional, repeated`
  indented lines
- `else :             # else optional`
  indented lines
- Python executes the first section that has a `True` condition, or else the `else` statement if present

Strings: convert to string, and manipulate them

- `str(x)` → default string representation of `x`
- `"banana".index('an') → 1 # first location`
- `"banana".index('an', 2) → 3 # location after 2`
- `"%".join(['a', 'b', 'c']) → 'a%b%c'`
- `"abcb".replace('b',"xx") → 'axxcxx'`
- `"a bc d ".split() → ['a', 'bc', 'd']`
- `"a bc d ".split('b') → ['a ', 'c d']`
- `"  ab c ".strip() → 'ab c'`
- `"'ab'c'".strip("'") → "ab'c"`
- `"200 Hike!".upper() → '200 HIKE!'`
- `"200 HIKE!".lower() → '200 hike!'`

File reading from file in current directory

-
  ```
  from os import chdir # change folder function
  from sys import path # current path
  chdir(path[0]) # change folder to .
  flink = open(filename, "rU") # open
  text = ""        # initialize input
  for each in flink :    # read line
      text += eachline   # save line
  flink.close()          # close file
  ```

File reading from url

-
  ```
  import urllib
  url = "http://sample.org/file.txt"
  flink = urllib.urlopen(url)
  # continue as when reading file
  ```

Numpy arrays

`import numpy as np`

A collection of ***same-type*** objects, with functions:

- `np.arange(a,b,c)  # like range`
- `np.linspace(0.1,0.8,n) # n floats`
- `np.zeros(n) # n zero floats`
- `np.array(x) # convert list or tuple x`
- `np.random.random(n) # n values in [0,1)`
- `np.random.randint(a,b,n) # int in [a,b)`
- `np.random.seed(s) # reset seed to s`
- `np.random.shuffle(x) # shuffle x`

Math operations with arrays:

- `+ - * / // % **` between two arrays or an array and a single value are done *item by item*
- `np.int_(x) # casts to int`
- *vectorized* math functions (e.g., `np.sqrt`, `np.sum`) handle real or complex array inputs
- `n1 = np.arange(5); n2 = n1[1::2]`
- `n2[:] = 4; n1 → [0, 4, 2, 4, 4]`

Plotting

- `import matplotlib.pyplot as plt`
- `fig = plt.figure(n) # which figure to alter`
- `fig.add_subplot(326,aspect="equal")`
  `# 3x2 rows, columns, 6'th subplot`
- `plt.plot(x,y,fmt,label="…") # plots y vs x`
  like `scatter`, `semilogx`, `semilogy`, `loglog`
- `fmt` (optional) is a string with colour and type:
  —`r` (red), `g` (green), `b` (blue), `k` (black)
  —`o` (circle), `-` (solid), `--` (dashed), `:` (dotted)
  —`<`,`v`,`>` (triangles), `s` (square), `*` (star)
- `plt.xlabel`, `plt.ylabel`, `plt.title`, `plt.legend("upper left")` add text to plot
- `plt.savefig` makes image file of plot
- `plt.hist(xx, 20)` plots histogram in 20 bins
- `plt.xlim(min, max)` sets limits on x-axis
- `plt.ylim(min, max)` sets limits on y-axis
- `plt.show()` makes the plot appear on the screen