THE UNIVERSITY OF MANITOBA

COMP1012: Computer Programming for Scientists and Engineers Final Exam (3 hours)

Name	
Student number	

Section (please check one):

☐ A01 (Andres TuTh) ☐ A03 (Amiri TuTh)

☐ A02 (Melvin MWF)

April, 2015

MARKS

Exam Instructions:

- Marks add up to 50.
- No aids are permitted.
- Answer all questions, and write your answers on the exam itself.
- Write your name, student number, and section on this page and any ***separated*** pages.
- Place your student card on your desk.

For Graders Only:		
A:	_______	/ 10
B:	_______	/ 17
C:	_______	/ 8
D:	_______	/ 10
E:	_______	/ 5
Total:	_______	/ 50

Part A: Predict the output (10 × 1 MARK)

There is a separate problem in each row of the table below. In each one, mentally execute the code fragment on the left and enter the expected output in the box on the right. ***None result in an error.*** Use the last page of the exam for scrap work.

	Code Fragment	*Expected output*
1. [1]	`print 3**2 + 2. // 4`	9.0 [order of operations, types]
2. [1]	`print sorted(list(set("banana")))`	['a', 'b', 'n'] [tuple operations]
3. [1]	`print 1 / 3 or 5**2`	25 [Boolean expressions]
4. [1]	`print range(0,0,1)`	[] [3-argument range]
5. [1]	`print (2, 0, 1, 5)[-2:]`	(1, 5) [slice of a tuple, neg. index]
6. [1]	`print [jj * 2 for jj in 'bat' ]`	['bb', 'aa', 'tt'] [list comprehension, repeat]

7. [1]	`print array([3, 2, 3, 2])[`
	`        array([0, 1, 1, 0]) == 0]`
		array([3,2]) or [3 2]
		[bool array indexing]

8. [1]	`DL=U'\N{BOX DRAWINGS HEAVY DOWN AND LEFT}'`
	`print DL`
		┓
		String processing; accept ┗ without penalty

9. [1]	`print 2 + 1j**2`	1+0j [complex number; accept 1. + 0.j or any combo]
10. [1]	`print {2, 4, 3}.intersection({1, 2})`	{2} [sets] could be written set([2])

Name

MARKS

Part B: Programming – (16 MARKS)

B.1 through B.3 together make up a program; you may want to look them all over.

B.1 Function chooseOne (4 marks)

[4] Define a function `chooseOne` that prompts the user over and over to pick an entry from a list of short choices until the user makes a valid choice. The choices come from the single parameter, `entryList`. If `entryList` is `["brown", "green", "yellow", "red"]`, the interaction might look like this:

```
Choose one of the following, by number:
1) brown  2) green  3) yellow  4) red
Choice: 6

Invalid. Enter a number from 1 to 4:
Choice: 2
```

Details:

- Display the choices only once, written across the line, with a number beside each.
- Numbering starts at 1.
- Check only the first non-blank character the user enters.
- Return the actual choice as a string (e.g., 'green'), not the number.
- No doc string required.

```
def chooseOne(entryList) : # start coding on next line

    """Display the parameter entries once, across the
    line, numbered, then solicit the user to pick one.
    Check only first character of response; must be
    valid digit."""

    print "Choose one of the following, by number:"
    for pos, entry in enumerate(entryList) :
       print "%d) %s\t" % (pos+1, entry),
    warn = '\b'
    while warn :
       userIn = raw_input(warn + "Choice: ").strip()
       warn = ''
       if not (userIn[0].isdigit()
          and '1' <= userIn[0] <= str(len(entryList))) :
          warn = ("Invalid. Enter a number from 1 to %s\n"
                 % len(entryList))
    return entryList[int(userIn[0]) - 1]

A: [1] Display choices
B: [1] Loop over input
C: [1] Check input
D: [1] Return result
```

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

Name

MARKS

B.1 through B.3 together make up a program; you may want to look them all over.

B.2 Function fnc3 (6 marks)

[6] The exponential series, sine series and cosine series have similar terms:

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

$$\cos x = 1 - \frac{x^2}{2!} + \frac{x^4}{4!} - \frac{x^6}{6!} + \dots$$

$$\sin x = \frac{x}{1!} - \frac{x^3}{3!} + \frac{x^5}{5!} - \frac{x^7}{7!} + \dots$$

Write one function `fnc3` with parameter *x* that returns (exp(*x*), cos(*x*), sin(*x*)). Details:

- Do not call any library functions. Evaluate the functions by summing these series.
- Evaluate each term only once and use it in the two series where it is needed.
- Do ***not*** calculate numerator and denominator separately.
- Discard terms with magnitudes less than 10^{-18}, and only those terms.
- No input nor output in this function.

```
def fnc3(xx) :
    """Generate (exp(xx),cos(xx),sin(xx)) using series, discarding
    terms less than 1.e-18."""
    count = 0
    sinTot = 0.
    expTot = 0.
    cosTot = 0.
    term = 1.
    sinSign = 1
    cosSign = 1
    while abs(term) >= 1.e-18 :
        expTot += term
        count += 1
        if count % 2 == 1 :
            cosTot += term * cosSign
            cosSign = -cosSign
        else :
            sinTot += term * sinSign
            sinSign = - sinSign
        term *= xx / float(count)
    return (expTot, cosTot, sinTot)

A: [2] Initialization
B: [1] Loop
C: [2] Update totals, count
D: [1] Return
```

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

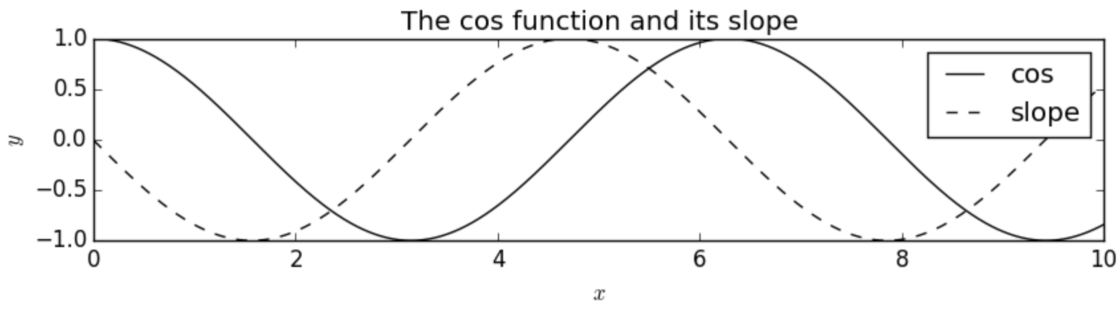
Name

MARKS

B.1 through B.3 together make up a program; you may want to look them all over.

B.3 Function main (7 marks)

[7] Define a function `main` that determines from the user which function to plot, either exp, cos or sin, by calling `chooseOne` from B.1. Your source of values for the chosen function is `fnc3` from B.2, which does ***not*** accept an array argument. Plot the required function from 0 to 10, with x values spaced 0.01 apart. Also plot the slope between consecutive y values $(y_{i+1} - y_i)/(x_{i+1} - x_i)$ as a function of x_i. Label the plot completely. Use equal aspect ratio. Shown is an example for cos.



```
def main :
    """Plot a selected function and its slope."""

    fncs = ["exp", "cos", "sin"]
    choice = chooseOne(fncs)
    which = fncs.index(choice)
    xs = np.linspace(0,10,1001)
    values = [fnc3(xx)[which] for xx in xs]
    ys = np.array(values)
    diffX = xs[1:] - xs[:-1]
    diffY = ys[1:] - ys[:-1]
    fig = plt.figure()
    fig.add_subplot(111,aspect="equal")
    plt.plot(xs, ys, 'k-', label=choice)
    plt.plot(xs[:-1],diffY / diffX, 'k--', label="slope")
    plt.legend(loc="best")
    plt.xlabel(r'$x$')
    plt.ylabel(r'$y$')
    plt.title("The %s function and its slope" % choice)
    plt.show()

A. [2] Generate x,y values for the chosen function
B. [2] calculate slopes, plot curves
C. [1] figure, subplot
D. [1] plot title, x, y axis labels
E. [1] plot legend, show
```

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

Name

MARKS

Part C: Programming – (8 MARKS)

C.1 Calculating Statistics [3 marks]

[3] Define a function `calcStats` that calculates and returns the mean and standard deviation of n x values. It should use the following formulas, where Σ means a summation:

$$\bar{x} = \frac{1}{n}\sum_{i=0}^{n-1} x_i$$

$$s_x = \sqrt{\frac{1}{n-1}\sum_{i=0}^{n-1}(x_i - \bar{x})^2}$$

Assume without checking there are n x-values, x_0 to x_{n-1}, and that $n > 1$. You may use `numpy`, or not, as you prefer.

```
import _________________
def calcStats(xs) :
    """Calculate and return the mean and standard deviation of the
    values in xs. Assume there are at least two values."""

    nn = float(len(xs))
    xBar = np.sum(xs) / nn
    resids = xs - xBar
    stdDev = np.sqrt(np.sum(resids * resids) / (nn - 1.))
    return xBar, stdDev
```

A. [1] importing Random and generating random character
B. [1] branching for mirrored
C. [1] loop for the mirrored triangle
D. [1] loop for the non-mirrored triangle
E. [1] defining a text variable to be returned as the triangle and updating it

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

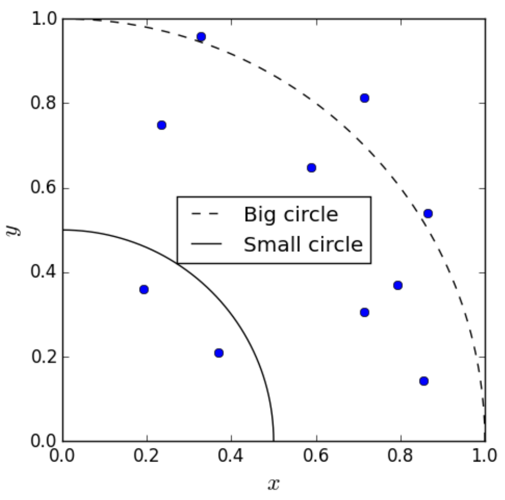
Name

MARKS

C.2 Estimate Area Ratio [5 MARKS]

[5] Write a program (no function definitions) to estimate the following ratio using the Monte Carlo method:

$$Ratio = \frac{\text{area of the small circle}}{\text{area of the big circle}}$$



As shown in the figure, the radii of the small and big circles are 0.5 and 1, respectively. Generate random points (x, y) where x and y lie between 0 and 1, then count the number of points that are inside the circles as a representative of their area. (Hint: (x, y) is inside a circle of radius r if $x^2 + y^2 \le r^2$) Ten points are shown in the figure as an example: 5 are just in the outer circle, and 2 are in both. The ratio 2 / 7 gives 0.28571.

Your program should generate output as follows:

```
Ratio estimate with      10 points is :   0.28571
Ratio estimate with    1000 points is :   0.25254
Ratio estimate with   1e+05 points is :   0.24920
Ratio estimate with   1e+07 points is :   0.25027
```

```
import numpy as np
# Do not import any library other than numpy.


rBig, rSmall = 1.0, 0.5
rBigSq, rSmallSq = rBig**2, rSmall**2
numPts = 10
while numPts <= 10**7 :
    xx = np.random.random(numPts)
    yy = np.random.random(numPts)
    distSq = xx**2 + yy**2
    ratio = ( np.sum(distSq < rSmallSq) /
        float(np.sum(distSq < rBigSq)) )
    print ("Ratio estimate with %7.4g points is : %9.5f"
           % (numPts, ratio)
    numPts *= 10

A. [1] while loop
B. [1] generation of the random x and y
C. [1] counting the points inside the big circle and small
   circle
D. [1] calculation of the ratio
E. [1] printing in the correct format
```

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

Name

MARKS

Part D: Multiple Choice + Expression (10 x 1 MARK)

For each of the following nine multiple-choice questions, circle the ***single best*** answer.

[1] 1. Given that `phrase` refers to a string with several words in it, which of the following is a valid Python statement?

a) `newPhrase = phrase.split().upper().strip().replace(',', ' ')`
[✓ b) `newPhrase = phrase.upper().strip().replace(',', ' ').split()`]
c) `newPhrase = phrase.strip().replace(',', ' ').split().upper()`
d) `newPhrase = phrase.replace(',', ' ').split().upper().strip()`
e) Both b) and c) are valid.

[1] 2. Given the following list comprehension, which of the expressions below has the value 3?

```
list1 = [[item] + [item * 2] for item in range(4)]
```

a) `list1[1]`
b) `list1[2]`
c) `list1[1][0]`
d) `list1[2][0]`
[✓ e) `list1[3][0]`]

[1] 3. Which of the following format specifiers could result in the given output?

```
3.450e-05
```

[✓ a) `%.3e`]
b) `%.3f`
c) `%.3g`
d) `%.3d`
e) both `a)` and `c)`

[1] 4. Which of these will NOT make `numbs` refer to a collection of four pseudo-random numbers between 0 and 1?

a) `numbs = np.random.random(4)`
b) `numbs = [random.random() for item in range(4)]`
c)
```
numbs = []
for item in range(4): numbs += [random.random()]
```
d)
```
numbs = np.array([random.random(), random.random(),
                  random.random(), random.random()])
```
[✓ e) `numbs = random.randrange(4)`]

[1] 5. Given the following sequence of statements, what would the output be?

```
def calc(aa, bb) :
    aa = 2
    return aa * bb

aa, cc = 5, 3
bb = 2
cc = calc(aa, bb)
cc = calc(cc, aa)
print cc
```

a) 6
[✓ b) 10]
c) 15
d) 30
e) 60

Name

MARKS

[1] 6. Which of the following statements is ***NOT*** correct?

a) `xx[3:17]` is a slice, whether `xx` is a list, a tuple, a string or a numpy array.
b) If `xx` is a numerical numpy array, data values in `xx[3:17]` are stored in contiguous memory locations.
c) This assignment works whether `xx` is a list or a numerical numpy array:
`xx[3:17] = np.arange(14)`
d) This assignment works whether `xx` is a list or a numerical numpy array:
`xx[3:17] = 14`
e) This assignment works if `xx` is a numerical numpy array:
`xx[xx < 5] = xx[xx < 5]**2`

[1] 7. Which of the following **cannot** be used as the stopping rule for evaluation of power series?

a) Stopping after term stops changing.
b) Stopping after a specific number of iterations.
c) Stopping when the total stops changing.
d) Stopping after the absolute value of term gets very small.
e) All of the above can be used.

[1] 8. In evaluating the expression below, which will be the last two operators evaluated?

```
1**5 or True + 5. / 6 // 6
```

a) Last: `or`, 2nd last: `**`
b) Last: `+`, 2nd last: `//`
c) Last: `or`, 2nd last: `+`
d) Last: `//`, 2nd last: `/`
e) Last: `+`, 2nd last: `or`

[1] 9. What is the output of the below program?

```
import numpy as np
var1 = np.linspace(1,5,6)
var2 = var1
var1[-1] = 10
print var2
a) [  1.   1.8   2.6   3.4   4.2   5.]
b) [  1.   1.8   2.6   3.4   4.2   10.  ]
c) [  1.   2.25  3.5   4.75  10.  ]
d) [  1.   2.25  3.5   4.75  6.   ]
e) [  1.   2.    3.    4.    5.    10.  ]
```

[1] 10. [1 marks] Using good coding practices and the same rules as QuizMaster write a Python expression to evaluate this mathematical expression, assuming `math` has already been imported:

$$\sqrt{\ln\left(\log_{10}(10)\right) \cdot b \cdot \frac{\pi}{5+3}}$$

Put expression here

```
short form: 0., since log10(10) is 1, and ln(1) is 0
long form:

math.sqrt(math.log(math.log10(10.)) * bb * math.pi / 8.)
```

Name

MARKS

Part E: Short Answer (5 MARKS)

[5] The code below counts the number of times each word appears in a list of words. It returns a list of pairs (word, count), where each pair has a different word. The code was originally correct, but 6 errors have been added to it. Each error is one of the following: ***missing/extra/incorrect item, where item is punctuation, constant, operation, library, function or variable*** (including mismatched brackets or quotes). Only numbered lines have errors, and no line has more than one error. ***The comments are correct.*** Find the errors. In the table at the bottom, give the line number of each error, say what is wrong and how you would fix it. As an example, one error has been done for you. Find five more.

```
   #..............................................................countWords
1  def countWords(words)
       """Analyze a list of words, and return a list of (word, count) pairs."""
2      wordDict = ()
3      for word in words :
4          if not word : # ignore empty words
5              if word in wordDict :
6                  wordDict[word] = 1
7              else :
8                  wordDict(word) = 1
9     return words.items()
```

(1) line 1... ':' is missing in the **def** statement; add it after the closing parenthesis ')'
(2) line 2: () should be {} to define a dictionary, not a tuple
(3) line 4: delete the not; ignore empty words, not full ones
(4) line 6: = should be += to increase the count
(5) line 8: () should be []
(6) line 9: wordDict.items, not words.items

Name

MARKS

Extra space

Name

MARKS

Extra space