THE UNIVERSITY OF MANITOBA

Name	
Student number	

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

Section (please check one):

☐ A01 (Andres TuTh)

☐ A02 (Boyer MWF)

December 11, 2014

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:	_____	/ 16
C:	_____	/ 12
D:	_____	/ 7
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**1 + 2. // 4`	`3.0` `[order of operations, types]`
2. [1]	`print tuple(("cattle",))`	`("cattle",)` `[tuple operations]`
3. [1]	`print 0**3 <= 3 and 3`	`3` `[Boolean expressions]`
4. [1]	`print range(-6,-2,2)`	`[-6,-4]` `[3-argument range]`
5. [1]	`print [0, 1, 2, 3][1:-1]`	`[1,2]` `[slice of a list, neg. index]`
6. [1]	`print [jj - 1 // 3. for jj in [5, 1, 3, 1] ]`	`[5., 1., 3., 1.]` `[slice of a tuple, divide]`
7. [1]	`print array([4, 0, 0, 2])[ array([0, 1, 2, 3]) <= 1]`	`array([4,0]) or [4 0]` `[bool array]`
8. [1]	`print "Thursday ".strip().upper( ).replace("THURS", "FRI")`	`FRIDAY` `String processing`
9. [1]	`print 2 + 1j**2`	`1+0j` `[complex number]`
10. [1]	`print {1, 2}.union({2, 3}) - {1, 4}`	`{2,3}` `[sets]`

Name

MARKS

Part B: Programming – (16 MARKS)

B.1 through B.4 together make up a program; you may want to look them all over before starting any of them.

B.1 Function `read4Words` (4 marks)

[4] Define a function `read4Words` that prompts the user over and over to enter four words, until the user enters four words, separated by blanks, all on one line. Return a list of the four words, all converted to lowercase, to the calling code. For example, in the session shown below, read4Words returns `["to", "be", "or", "not"]`.

```
Enter 4 words on one line, separated by blanks: first word

Your input had 2 words; try again.
Enter 4 words on one line, separated by blanks: I am trying!

Your input had 3 words; try again.
Enter 4 words on one line, separated by blanks: To be or NOT
```

```
def read4Words() :
    """Prompt the user to enter 4 words on a line, separated by
    blanks. Repeat until success."""

    PROMPT = "Enter 4 words on one line, separated by blanks: "
    warning = "\n"
    while warning :
        userInp = raw_input(warning + PROMPT)
        warning = ""
        words = userInp.split()
        if len(words) != 4 :
            warning = ("Your input had %d words; try again.\n"
                       % len(words))
    return [word.lower() for word in words]
```

A: [2] Loop and input
B: [2] Break into words; return in lower case

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

Name

MARKS

B.1 through B.4 together make up a program; you may want to look them all over before starting any of them.

B.2 Function allCombos (4 marks)

[4] Write a function `allCombos` that, given a list of four words, returns a list of all strings with one letter from each word. The first letter should come from the first word, the second letter from the second word, etc. Sort the list of strings into alphabetical order. For example, from `['too', 'many', 'items', 'too']`, produce 180 words: `['oaeo', 'oaeo', 'oaeo', …, 'tytt']`. Do not remove duplicates.

```
def allCombos(words4) :
    """Generate all strings that can be made with the 1st character
    from the 1st word of words4, the 2nd character from the 2nd
    word, etc. Return a sorted list."""
    combos = ["".join([c0, c1, c2, c3])
                for c0 in words4[0]
                    for c1 in words4[1]
                        for c2 in words4[2]
                            for c3 in words4[3]]
    combos.sort()
    return combos
```

```
A: [2] Generate list of combos; could use nested list
comprehension, or nested for loops
B: [1] Sort the list.
C: [1] Return the list
```

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

MARKS

B.1 through B.4 together make up a program; you may want to look them all over before starting any of them.

B.3 Function filterWords (4 marks)

[4] Define a function `filterWords` that finds all strings in a list that are words from a given wordlist. It returns a ***tuple*** containing unique words sorted into increasing alphabetical order. For example, given the set of 180 combos produced by `allCombos` for the input `['too', 'many', 'items', 'too']`, and a wordlist containing all common English words, `filterWords` returns `('omit', 'onto')`.

```
def filterWords(combos, wordList) :
    """Find all strings in combos that are words in wordList,
    and return a sorted tuple without duplicates."""

    words = list(set(combos).intersection( set(wordList) ))
    words.sort()
    return tuple(words)
```

```
A. [2] Find unique items; easy with set, or dict, harder with
   list.
B. [1] sort a list (can't sort a set or a tuple)
C. [1] convert to tuple and return
```

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

MARKS

B.4 Function `main` [4 MARKS]

[4] Define a function `main` that uses the functions on the previous pages to find all the common English words that can be made by taking one letter from each of four words provided by the user. It should call each of `read4Words`, `allCombos` and `filterWords` exactly once. The program as a whole should produce output very similar to the sample below. Note the spacing. Assume there will be at least seven strings, and show the first five and the last one, separated by "...". In contrast, show *all* the unique words.

```
Enter 4 words on one line, separated by blanks: Handle equipment
with care!

User's words:   handle  equipment  with  care!

Sorted strings: aeh!  aeh!  aeha  aeha  aehc  ...  nuwr

Unique words:   amir  ante  emir  epic  heir  lite  lute  nite
```

```
def main() :
    """Find all words that can be made from 4 given words."""
    wordList = getWords("2of12inf.txt") # do NOT write getWords

    words = read4Words()
    print "\nUser's words: \t%s" % "  ".join(words)
    combos = allCombos(words)
    print "\nStrings: \t%s" % "  ".join(
                            combos[:5] + ["..."] + combos[-1:])
    uniqueWords = filterWords(combos, wordList)
    print "\nUnique words: \t%s" % ("  ".join(uniqueWords))
    return
```

A: [2] Call read4Words, allCombos, and filterWords, passing correct arguments and saving return values
B: [2] Print the lines of output, showing the correct values and correct formatting.

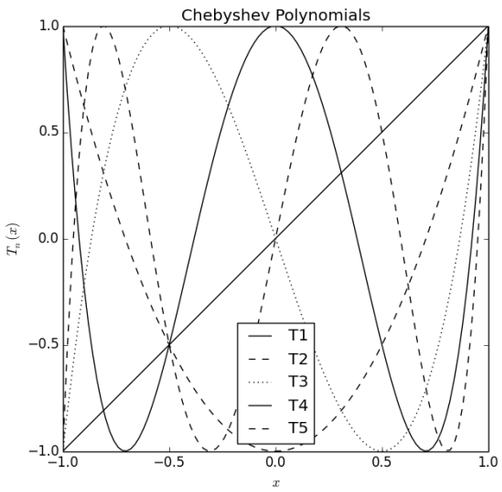
For marker use only	
Item	Mark
A	
B	
C	
D	
E	
Sum	

Name

MARKS

Part C: Programming – (12 MARKS)

C.1 Chebyshev Polynomials [6 MARKS]



[6] Write a script (it doesn't have to have any input nor any functions) to plot the Chebyshev polynomials $T_n(x)$ for n varying from 1 to 5. Use a loop over values of n. Use 101 equally spaced x values from -1 to 1. Do not use any loops related to x; use numpy array calculations instead. Use black as your colour for all curves, and use solid, dashed and dotted lines. Label the plot as shown. The following definition enables you to determine $T_n(x)$.

$$T_n(x) = \begin{cases} 1 & \text{if } n = 0 \\ x & \text{if } n = 1 \\ 2xT_{n-1}(x) - T_{n-2}(x) & \text{otherwise} \end{cases}$$

```
# no leading comments are required
import matplotlib.pyplot as plt
import numpy as np

xs = np.linspace(-1, 1, 101)
Tpoly = [0 * xs + 1, xs]

for num in range(2,6) :
    Tpoly.append(2 * xs * Tpoly[-1] - Tpoly[-2])

styles = ["-k", "--k", ":k"]
fig = plt.figure(1)
plt.clf()
fig.add_subplot(111, aspect="equal")

for count, array in enumerate(Tpoly[1:]) :
    plt.plot(xs, array, styles[count % len(styles)],
                                    label = "T%d" % (count+1) )
plt.title("Chebyshev Polynomials")
plt.xlabel("$x$")
plt.ylabel("$T_n(x)$")
plt.legend(loc="best")
plt.show()
```

A. [2] Generate xs and function values.
B. [2] figure and plot calls; add_subplot not required
C. [2] xlabel, ylabel, title, legend, show; Latex formats not required

They can use functions, but must call them in a script:
D. -0.5 if they do not.

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

Name

MARKS

C.2 Simulate Dice [6 MARKS]

[6]

Write a script with a single function `main` that uses 1000 simulations to estimate the probability that *m* takes the value 6, where *m* is defined as follows:

- roll a single fair 6-sided die repeatedly, for as long as the result is an odd number;
- let *r* be the number of rolls until the first even number: if *r* is divisible by 3, let *m* be the sum of the *r* rolls; otherwise let *m* be the value of the last roll.

Use at least two global constants. Set the random seed to 2014. Sample output:

```
Based on 1000 simulations, the probability that m = 6 is 0.298

Programmed by <your name>
```

```
# no leading comments are required; use at least two constants
# define a main function
import numpy as np
import random

TARGET = 6
NUM_ROLLS = 1000

def main() :
    random.seed(2014)
    count = 0
    for trial in range(NUM_ROLLS) :
        rolls = 0
        dieVal = 7  # artificial value to start
        total = 0
        while dieVal % 2 == 1 :
            rolls += 1
            dieVal = random.randrange(1,7)
            total += dieVal
        if rolls % 3 == 0 :
            mm = total
        else :
            mm = dieVal
        count += mm == TARGET
    fmt = ("Based on %s simulations, the probability that "
           "m = %s is %g")
    print fmt % (NUM_ROLLS, TARGET, float(count)/NUM_ROLLS)

    print "\nProgrammed by <your name>"
    return
main()
```

A. [1] correct use of main
B. [1] correct use of random to generate dice rolls, including seed
C. [1] generating 1000 simulations, counting successes
D. [2] generating value of m
E. [1] output

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

MARKS

Part D: Multiple Choice + Expression (7 x 1 MARK)

For each of the following 6 multiple-choice questions, circle the ***single best*** answer.

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

```
list1 = range(5)
list2 = [list1, list1, 3]
list1[0] = list2
```

a) `list1[2]`
b) `list2[1][2]`
c) `list1[0][0][2]`
d) `list2[0][0][0][2]`
e) `list1[0][0][0][2]`

[1] 2. Which of the following format specifiers can be used to print a complex number?

a) `%c`
b) `%d`
c) `%f`
d) `%g`
e) `%r`

[1] 3. Considering the following sequence of instructions, what is the result of `draw(5, '@')`? `[answer: b]`

```
def draw(SIZE, CHAR) :
    for row in range(SIZE + 1) :
        text = ""
        for col in range(min(row + 1, SIZE)) :
            text += CHAR
        print text
```

a)
```
@@@@@
@@@@@
@@@@
@@@
@@
@
```

b)
```
@
@@
@@@
@@@@
@@@@@
@@@@@
```

c)
```
@
@@
@@@
@@@@
@@@@@
```

d)
```
@@@@@
@@@@
@@@
@@
@
```

e)
```
@@@@@
@@@@@
@@@@@
@@@@@
@@@@@
```

[1] 4. Which of the following statements is ***not*** correct?

a) Lists, tuples, strings and numpy arrays all use the same notation for slices.
b) A string is like a tuple of characters, in that both are immutable.
c) Among lists, tuples, strings and numpy arrays, only tuples ***cannot*** be concatenated with the '+' operator.
d) One difference between a list and a numpy array is that data items in the array all have the same data type.
e) Among lists, tuples, strings and numpy arrays, you can subtract 5 only from an array.

MARKS

[1] 5. Which of the formatted prints below would produce this output?

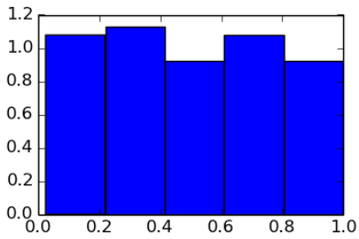
```
Output: |-2.8571e-01          |
```

a) `print "Output: |%-20.4e|" % (-2./7)`
b) `print "Output: |%-20.4f|" % (-2./7)`
c) `print "Output: |%-20g|"   % (-2./7)`
d) `print "Output: |%20.4f|"  % (-2./7)`
e) `print "Output: |%20.4e|"  % (-2./7)`

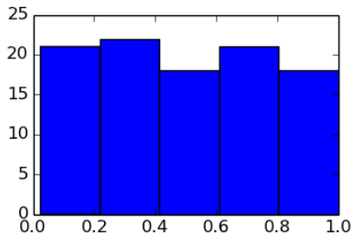
[1] 6. Which plot could result from the following code? [answer: d]

```
import numpy as np
import matplotlib.pyplot as plt
plt.figure()
numbers = np.random.random(100)
plt.hist(numbers)
plt.show()
```

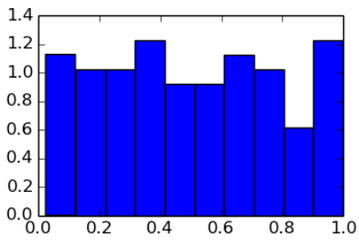
a)



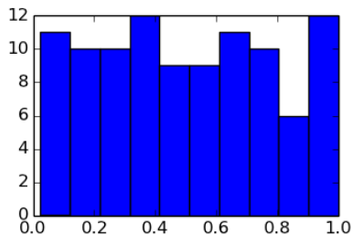
b)



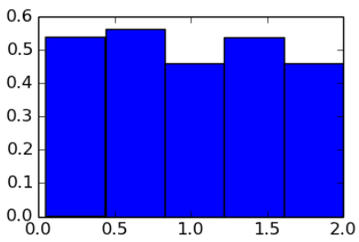
c)



d)



e)



[1] 7. [1 marks] Using good coding practices and QuizMaster rules, write a Python expression for this expression, assuming `math` has already been imported:

$$\left\lceil \frac{\sqrt{a}}{-e^{\tan(10)}\pi^{\frac{10}{\lfloor x \rfloor}}} \right\rceil$$

Put expression here

```
math.ceil(math.sqrt(aa) / (-math.exp(math.tan(10.)) * math.pi**(10. /
    math.floor(xx)))
```

Name

MARKS

Part E: Short Answer (5 MARKS)

[5] The code below applies the sieve of Eratosthenes to find and return all the prime numbers up to a limit. 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.

```
#.............................................................findPrimes
1   def findPrimes(limit)
        """Find all prime numbers up to integer limit > 1 and return as array"""
2       mayBePrime = np.zeros(limit + 1) != 0  # limit + 1 Trues
3       mayBePrime[2] = False                # 2 Falses at the start
4       maxFactor = int((limit + 1)**0.5)    # largest small factor up to limit
5       for candidate in range(2, maxFactor) :
6           if mayBePrime[candidate] :
7               mayBePrime[2 * candidate : candidate : ] = False
            # all composites up to limit have False in mayBePrime, as at least
            # one of their prime factors has been found. The Trues are prime.
8       primes = np.range( len(mayBePrime) )[mayBePrime]
9       return
```

(1) line 1... ':' is missing in the **def** statement; add it after the closing parenthesis ')'

(2) `line 2: != 0 should be == 0`

(3) `line 3: mayBePrime[2] should be mayBePrime[:2]`

(4) `line 7: slice should be [2 * candidate : : candidate]`

(5) `line 8: np.range should be np.arange`

(6) `line 9: return should return the entire primes array`

MARKS

Extra space

MARKS

Extra space