THE UNIVERSITY OF MANITOBA

Name	
Student number	

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

Section (please check one):

☐ A01 (Amiri Armes 208)

☐ A02 (Andres Drake 343)

April 24, 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:	_____	/ 9
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 5 % 1**3 * 2`	`0` `[order of operations]`
2. [1]	`print list("too")`	`['t', 'o', 'o']` `[list operations]`
3. [1]	`print 3 >= 3 != 2 + 2`	`True` `[Boolean expressions]`
4. [1]	`print range(1,0,-2)`	`[1]` `[3-argument range]`
5. [1]	`print [0, 1, 2, 3][-4:1]`	`[0]` `[slice of a list, neg. index]`
6. [1]	`print ('Sun', 'Mon', 'Tue', 'Wed', 'Thu', 'Fri', 'Sat')[-7:4:2]`	`('Sun', 'Tue')` `[slice of a tuple]`
7. [1]	`print [jj // 3 / 1. for jj in [4, 2, 2, 4] ]`	`[1.0, 0.0, 0.0, 1.0]` `[list comprehension]`
8. [1]	`print array([0, 4, 3, 1])[array([0, 1, 2, 3]) == 1]`	`array([4])` `[array processing]`
9. [1]	`print "1".join(['1', '3', '5'])`	`"11315"` `[string function – quotes optional]`
10. [1]	`print 'COMP1012'.index('1012')`	`4` `[string function]`

Name

MARKS

Part B: Programming – (16 MARKS)

B.1 Function `nearMultiples` without numpy (4 marks)

Define a function `nearMultiples` that returns a list of all positive integers up to (but not including) a specified `limit` that are very close to multiples of the float parameter `number`. The small value `eps` defines how close they need to be. For example, given `limit = 100`, `number = 4.12` and `eps = 0.05`, it would return `[33, 70]`, since 8×4.12=32.96 is within 0.05 of 33 and 17×4.12=70.04 is within 0.05 of 70.

Write two versions of the function definition, one on this page that does ***not*** use `numpy`, and one on the next page that ***does*** use `numpy`, and contains ***no loops***.

[4]

```
# First version: function definition that does NOT use numpy
def nearMultiples(limit, number, eps) :
    """Return a list of all positive integers up to limit that are
    within eps of a multiple of number."""

    count = 0
    multiple = number
    nears = []
    while multiple < limit :
        count += 1
        multiple = count * number
        nearby = round(multiple)
        nears += [int(nearby)] * (abs(multiple - nearby) < eps)
    return nears
```

A: [2] Loop over candidates; limit applies to product
B: [2] Identify, accumulate, return successes; don't require math. prefix; need comparison above and below integer; convert result to int

If wrong values returned, deduct here, but allow them to simplify code in B.4 without penalty.

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

MARKS

B.2 Function using numpy (4 marks)

[4]

```
# Second version: definition that DOES use numpy, with NO LOOPS
import numpy as np
def nearMultiplesNp(limit, number, eps) :
    """Return a list of all positive integers up to limit that are
    within eps of a multiple of number."""

    maxFactor = int(limit // number)
    factors = np.arange(1, maxFactor + 1)
    multiples = factors * number
    nearInts  = np.int_(np.round(multiples))
    close = np.abs(multiples - nearInts) < eps

    return list(nearInts[close])
```

A: [2] Generate array of candidates without loop; list comprehension treated as loop; must use some numpy operations for any marks; -1 for loop if they do
B: [2] Identify, extract and return list of successes; must have np. prefix on functions; conversion to list not required

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

Name

MARKS

B.3 Function `getIntFloat` [4 MARKS]

[4] Define a function `getIntFloat(prompt)` that displays the `prompt` parameter and reads a line of input from the user. The input should contain an **`int`** followed by a **`float`**, separated by any number of spaces or tabs. The code must check only that there are two entries on the line. Assume without checking that they can be converted to an **`int`** and a **`float`** without error. Keep asking the user until two numbers are entered. Print a warning message for each invalid input, ***using the standard method taught in class***. Finally, return the **`int`** and **`float`** values. For example, here is a user interaction with one false start for the `prompt "Enter a and b: "`:

```
Enter a and b: 0.8

Invalid entry: 0.8

Enter a and b: 1000 0.8
```

```
def getIntFloat(prompt) :
    """Get an int and a float from the user on one line separated
    by spaces; check there are two values on the line."""

    warning = '\b'
    while warning :
        usrInp = raw_input(warning + prompt).strip().split()
        warning = ''
        if len(usrInp) != 2 :
            warning = ("Invalid input: %s \n\n"
                       % ' '.join(usrInp))
    return int(usrInp[0]), float(usrInp[1])
```

A. [1] input and split fields
B. [1] loop executes once, repeats on invalid input only; must use flag method, must reset warning
C. [1] test and warning message
D. [1] convert and return values

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 make a table of integers that are near multiples of the number π.

- It should call `getIntFloat` to find out which range of numbers to search, and how close they must be to integers.
- It should call `nearMultiples` (it doesn't matter which version) to find integers that are close to multiples of π.
- It should print a table of results. The entire interaction should look like this, including an abbreviated termination output:

```
INTEGERS THAT ARE NEAR MULTIPLES OF PI
Enter an integer limit and a small number (eg, 1000 0.1) 1000 .001

     Factor  Multiple of pi  Integer
  1    113      355.0000        355
  2    226      709.9999        710

Programmed by <your name>
```

```
def main() :
    """Print a table of pi multiples near integers."""

    print "Integers that are near multiples of pi".upper()
    prompt = ("Enter an integer limit and a small number "
              "(e.g., 10000 0.1) ")
    limit, eps = getIntFloat(prompt)
    target = math.pi  # So changing numbers is easy
    nears = nearMultiples(limit, target, eps)
    print "     Factor  Multiple of pi  Integer"
    count = 0
    for near in nears :
        factor = (near + 0.5) // target
        count += 1
        print "%3d %6d  %12.4f %10d" % (count,
              factor, factor * target, near)

    print "Programmed by <student name>"
```

A: [1] Input up to call to getIntFloat
B: [1] Multiples with call to nearMultiples
C: [2] Table: 0.5 each for headings, loop, printing line computing values to print

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

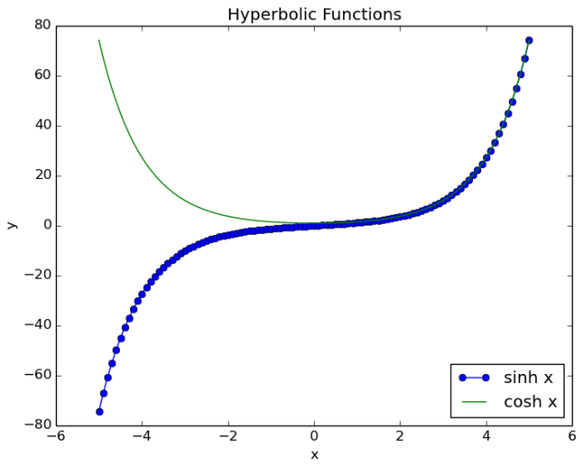
Name

MARKS

Part C: Programming – (9 MARKS)

C.1 Plot sinh(x), cosh(x) [6 MARKS]

[6]



Write a script (it doesn't have to have any input or any functions) to plot the hyperbolic sine and cosine functions from -5 to 5 using 101 points on each curve. Use `numpy` to avoid loops. Unfortunately, unlike `math`, `numpy` does ***not*** have a sinh function. However, you can use the following definitions and the `np.exp` function to evaluate them. Put a title on the plot and suitable labels, and include a legend. Put points on one of the curves to tell them apart in black and white printing.

$$\sinh(x) = \frac{e^x - e^{-x}}{2}$$

$$\cosh(x) = \frac{e^x + e^{-x}}{2}$$

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

xs = np.linspace(-5,5,101)
expx = np.exp(xs)
overExpx = 1. / expx
sinhx = 0.5 * (expx - overExpx)
coshx = 0.5 * (expx + overExpx)

plt.figure()
plt.plot(xs, sinhx, "o-", label="sinh x")
plt.plot(xs, coshx, label="cosh x")
plt.xlabel("x")
plt.ylabel("y")
plt.title("Hyperbolic Functions")
plt.legend(loc="best")
plt.show()
```

```
A. [2] Generate xs and function values.
B. [2] figure and plot calls
C. [2] xlabel, ylabel, title, legend, show

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 [3 MARKS]

[3] Write a script (it doesn't have to have any input or any functions) to estimate the probability that when you throw two fair 6-sided dice, the number of spots on the two dice differs by 1 (e.g., 4 and 3, or 1 and 2). Using either `random` or `numpy.random`, simulate throwing two dice 10000 times and estimate the probability from those results. Initialize the random seed first. Print the results like this:

```
Estimated probability two dice differ by 1 is 0.2791

Programmed by <your name>
```

```
# no leading comments are required
import numpy as np
import random

np.random.seed(2014)
NUM_ROLLS = 10000
dice1 = np.random.randint(1,7,NUM_ROLLS)
dice2 = np.random.randint(1,7,NUM_ROLLS)
diff = np.abs(dice1 - dice2)
count = np.sum(diff == 1)
print "Estimated probability two dice differ by 1 is %0.4f" % (
         float(count) / NUM_ROLLS )

print "Programmed by <student name>"

A. [1] generate dice, including seed
B. [1] count event
C. [1] calc and print result
```

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

MARKS

Part D: Multiple choice (10 x 1 MARK)

For each of the following 10 multiple-choice questions, circle the ***single best*** answer.

[1] 1. Which of the following formatted prints would produce the given output?

```
Solution : %  -6.000000e-01%
```

a) `print "%s : %%15.6e%%"  % ("Solution", -3./5)`
b) `print "%-s : %%15.6f%%" % ("Solution", -3./5)`
c) `print "%s : %%%15.6e%%" % ("Solution", -3./5)`
d) `print "%s : %%%15.3e%%" % ("Solution", -3./5)`
e) `print "%s : %15.6g%%"   % ("Solution", -3./5)`

[1] 2. What is meant by the Sieve of Eratosthenes?

a) A way of determining all prime numbers up to a limit.
b) A random number generator used by Python.
c) A rule describing the exponential growth of computer chip capacity.
d) A method for efficiently evaluating a polynomial.
e) The best way of summing up an infinite series.

[1] 3. Which one is ***not*** among the similarities of lists and arrays?

a) Operators **`is`** and **`in`** are applicable for both.
b) They use the same slice notation (`[a:b:c]`).
c) The contents of both can be changed.
d) Function `len` is applicable for both.
e) Operators * and + have the same effect on both.

[1] 4. After the following code executes what is the output?

```
import numpy as np
xArray = np.array([5, 10, 8])
yArray = np.array(xArray)
xArray[1] = 100
print yArray[:2]
```

a) `[  5  10]`
b) `[  5  10  8]`
c) `[  5  100  8]`
d) `[  5  100 ]`
e) `[  5]`

5. What is the value of `ss` when the following python code is run?

```
jj = 10; counter = 1
while jj :
   jj += counter
ss = jj
```

a) `46`
b) `ss` is not assigned a value.
c) `55`
d) `error`
e) `50`

[1] 6. Which of the following statements is correct?

a) Lists and tuples are immutable, but strings and `numpy` arrays can change.
b) Lists and `numpy` arrays are immutable, but tuples and strings can change.
c) Tuples and strings are immutable, but lists and `numpy` arrays can change.
d) Strings and lists are immutable, but tuples and `numpy` arrays can change.
e) Tuples, strings and `numpy` arrays are immutable; only lists can change.

MARKS

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

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

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

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

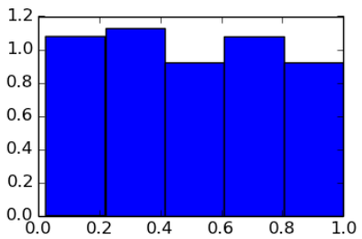
c)
```
@
@@
@@@
@@@@
@@@@@
```

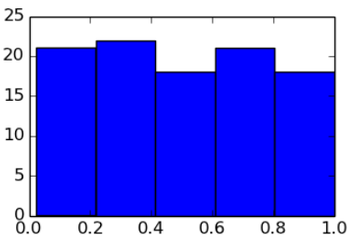
d)
```
@@@@@
@@@@
@@@
@@
@
```

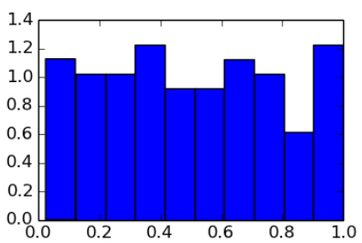
e)
```
@@@@@
@@@@@
@@@@@
@@@@@
@@@@@
```

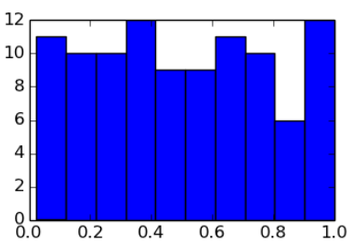
[1] 8. Which plot could result from the following code? `[answer: a]`

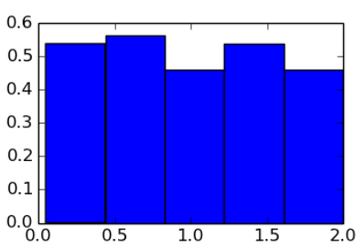
```
import numpy as np
import matplotlib.pyplot as plt
plt.figure()
numbers = np.random.random(100)
plt.hist(numbers, normed=True , bins = 5)
plt.show()
```

a)


b)


c)


d)


e)


MARKS

[1] 9. Which of the following is NOT a keyword of the Python language?

a) `None`
b) `elseif`
c) `while`
d) `assert`
e) `is`

[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:

$$x^{\lceil 3 \rceil} \cdot (y + \tan(x)) \cdot (10^{2+b}) .$$

Put expression here

```
xx**math.ceil(3.) * (yy + math.tan(xx)) * (10.**(bb + 2.))
```

MARKS

Part E: Short Answer (5 MARKS)

[5] The code below prints successive approximations to **pi** based on increasing numbers of terms of the arcsin(y) series, using the formula ($6 * \arcsin(0.5) = \pi$)

$$\arcsin(y) = (y) + \frac{1}{2}\left(\frac{y^3}{3}\right) + \frac{1\cdot3}{2\cdot4}\left(\frac{y^5}{5}\right) + \frac{1\cdot3\cdot5}{2\cdot4\cdot6}\left(\frac{y^7}{7}\right) + \frac{1\cdot3\cdot5\cdot7}{2\cdot4\cdot6\cdot8}\left(\frac{y^9}{9}\right) + \ldots$$

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. 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.

Sample output:
With 1 terms, estimate of pi = 3.00000
With 2 terms, estimate of pi = 3.12500
With 3 terms, estimate of pi = 3.13906
With 4 terms, estimate of pi = 3.14116
With 5 terms, estimate of pi = 3.14151

```
1   def piEstimator(xx , eps)
        "Return a list of arcsin(xx) terms"
2       count = 0 # number of terms so far
3       product = xx
4       terms = 0
5       term = xx
6       while abs(term) > eps :
7           count += 1
8           terms += [term]
9           product = xx**2 * (2. * count - 1.) / (2. * count)
10          term = product / (2. * count - 1.)
11          return terms

12  terms = piEstimator(0.5, 1.0e-5)
13  for num in range(1, len(terms)) :
14      print "With %d terms, estimate of pi = %.5f" % (num, 4.0 * sum(terms[:num]))
```

(1) line 1... ':' is missing in the **def** statement; add it after the closing parenthesis ')'
(2) `line 4: change terms = 0 to terms = []`
(3) `line 9: change the assignment operator from '=' to '*='`
(4) `line 10: change (2. * count - 1.) to (2. * count + 1.)`
(5) `line 11: return should be indented back to outside of the while loop`
(6) `line 14: replace 4.0 * sum(terms[:num]) with 6.0 * sum(terms[:num])`

MARKS

Extra space

MARKS

Extra space