THE UNIVERSITY OF MANITOBA

Name	
Student number	

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

Section (please check one):

☐ A01 (Andres Armes 208)

☐ A02 (Amiri Armes 201)

April 22, 2013, 9:00 am

MARKS

Exam Instructions:

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

For Graders Only:	
A:	_____/10
B:	_____/20
C:	_____/10
D:	_____/ 5
E:	_____/ 5
Total:	_____/50

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

There is a separate problem in each row of the table below. For 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 2 % 2**4 + 1.`	`3.0`
		
		`[order of operations, float]`

2. [1]	`print tuple(np.cumsum(`
	`  np.linspace(1,2,2)))`
		`(1.0, 3.0)`
		
		`[combined array operations, cast]`

3. [1]	`print 2 + 3 > 3 and 2`	`2`
		
		`[behaviour of and with non-booleans]`

4. [1]	`print range(4,0,-2)`	`[4,2]`
		
		`[3-argument range]`

5. [1]	`print [0, 1, 2, 3][-3:-2]`	`[1]`
		
		`[slice of a list, negative index]`

6. [1]	`print ('Sun', 'Mon', 'Tue', 'Wed', 'Thu', 'Fri', 'Sat')[3:3]`	`()`
		
		`[slice of a tuple]`

7. [1]	`print (array([1, 4, 0, 5]) * 3`
	`  - 2)`
		`array([1, 10, -2, 13]) or`
		`[1 10 -2 13] without commas`
		
		`[array operations with scalar]`

8. [1]	`print array([3, 2, 5, 0])[`
	`  array([0, 1, 2, 3]) <= 2]`
		`array([3, 2, 5]) or`
		`[3 2 5]`
		
		`[masking an array]`

9. [1]	`print [jj % 2. * 3 for jj in`
	`  [3, 5, 2, 1] ]`
		`[3., 3., 0., 3.]`
		
		`[list comprehension]`

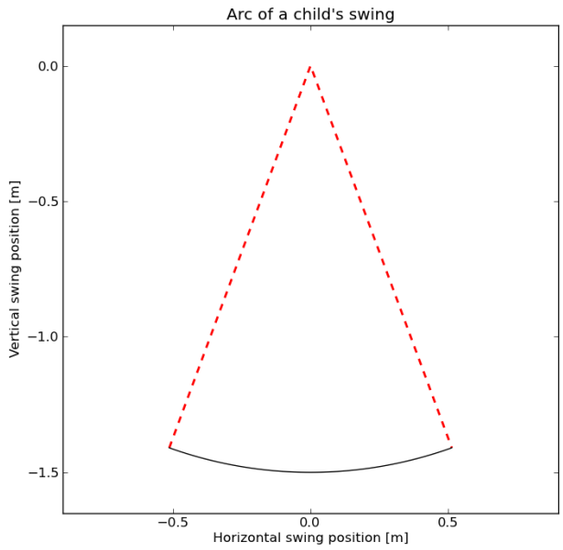
10. [1]	`print "-".join([ch for ch in`
	`  "apple"])`
		`"a-p-p-l-e"`
		
		`[string function, list comprehension]`

Part B: Programming – Full Program (4 + 4 + 3 + 4 + 5 = 20 MARKS)

Write a program that reads input about a child's swing, then prints the period of the swing in seconds and its maximum velocity in metres/second. Also, plot the arc of the swing. Here is the printed output from one run of the program:

```
Modelling A Child's Swing

Enter a value for the initial angle [degrees from vertical] 20

Enter a value for the length of support chain on swing [m] 1.5

Gravitational acceleration: 9.81 m/s2
Initial angle of swing:     20 degrees
                            0.349066 radians
Length of chains on swing:  1.5 metres

It takes 2.46 seconds to complete one swing
The maximum velocity of the swing is 1.33 m/s

Program terminating
```

The plotted output is to the right.



The main function has been programmed for you (below). Study it carefully. All of the interesting work done by the program is performed in the functions it calls. Over the next few pages you will write ***all*** of the underlined functions in `main` (no choice), and two other functions used at a lower level. Each function will be marked separately, so even if you have problems with one of them, you can still get full marks on others.

```
import math
import numpy as np
import matplotlib.pylab as plt

def main() :
   print "\nModelling A Child's Swing"

   GRAVITY, initialAngle, chainLength = getDetails()

   period, maxVel = pendulum(GRAVITY, initialAngle, chainLength)

   print "\nIt takes %.2f seconds to complete one swing" % period
   print "The maximum velocity of the swing is %.2f m/s" % maxVel

   plotSwing(GRAVITY, initialAngle, chainLength)

   print "\nProgram terminating"
```

Do NOT put your answer on this page.

Name

MARKS

B.1 Function `getDetails` [4 MARKS]

[4] Define a function `getDetails` that matches the call from `main`. It returns three values in the following order: the constant gravitational acceleration at the earth's surface of 9.81 m/s^2, the initial angle (in radians) at which the swing is released, and the length of the chains or ropes supporting the swing. The gravitational acceleration is a constant value, and ***the other two values are obtained from the user by calling*** **`getPosFloat`**. ***You may find it easier to go to the next page and write*** **`getPosFloat`** ***first, then come back and write*** **`getDetails`.**

The angle obtained from the user will be in degrees, and `getDetails` must convert it to radians. Function `getDetails` prints out these three values before returning them. Input in `getPosFloat` and output from `getDetails` should look like this:

```
Enter a value for the initial angle [degrees from vertical] 20

Enter a value for the length of support chain on swing [m] 1.5

Gravitational acceleration: 9.81 m/s2
Initial angle of swing:     20 degrees
                            0.349066 radians
Length of chains on swing:  1.5 metres
```

```
def getDetails() :
    """Return details about a child's swing: gravitational constant
    g, the initial angle at which it is released, and the length of
    the chain from the point of support."""

    GRAVITY = 9.81 # [m/s2] approximate acceleration due to gravity

    angleDeg = getPosFloat(
                    "the initial angle [degrees from vertical]")
    angleRad = angleDeg * math.pi / 180.

    chainLength = getPosFloat(
                    "the length of support chain on swing [m]")

    print "\nGravitational acceleration: %g m/s2" % GRAVITY
    print "Initial angle of swing:     %g degrees" % angleDeg
    print "                            %g radians" % angleRad
    print "Length of chains on swing:  %g metres" % chainLength

    return GRAVITY, angleRad, chainLength
```

Note: there is extra space on page 8

Name

MARKS

B.2 Function `getPosFloat` [4 MARKS]

[4] Define a function `getPosFloat` that matches the call from `getDetails`. It receives a single parameter value: `description`. It uses this parameter to create a prompt message that it uses to get input from the user. The input is a single float value that must be greater than 0.0. This function loops as many times as it takes to get a valid input from the user, which it returns as a float to the calling function.

Assume the user always enters a single float value, so you don't have to check for invalid input. You just have to check for the input's value being positive. If the user enters a negative or zero value, print a warning message before prompting again. For example, entry of the initial angle with a user mistake might look like this:

```
Enter a value for the initial angle [degrees from vertical] -20
The value -20 <= 0; enter a positive number

Enter a value for the initial angle [degrees from vertical] 20
```

```
def getPosFloat(description) :
    """Return a positive float value provided by the user when
    prompted with description. Loop if necessary to get acceptable
    input."""

    warning = "\n"
    while warning :
        userInput = raw_input(warning
                    + "Enter a value for %s " % description)
        value = float(userInput.strip())
        warning = ""
        if value <= 0.0 :
            warning = ("The value %g <= 0; "
                       "enter a positive number\n\n" % value)
    return value
```

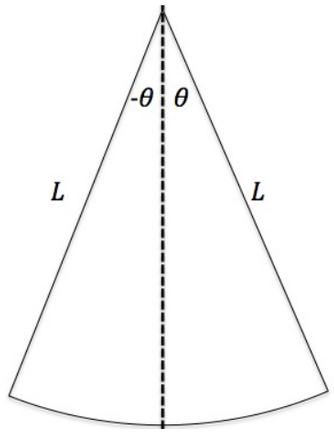
Note: there is extra space on page 8

MARKS

B.3 Function `pendulum` [3 MARKS]

[3]

Define a function `pendulum` that matches the call from `main`. It has three parameters, specifying the local gravitational acceleration, the initial angle of a simple pendulum and the radius of the arc of the pendulum's swing. (We will assume that a child's swing is approximately the same as a pendulum.)



This function returns two computed quantities: the period of the pendulum (how long it takes to return to its starting point), and the maximum velocity of the pendulum, which occurs when it is at its lowest point. The approximate period of a pendulum T is given by this equation:

$$T \approx 2\pi\sqrt{\frac{L}{g}}$$

where L is the length of the pendulum, and g is the gravitational acceleration.

The maximum speed can be determined from an energy argument. It is given by this formula:

$$V_{max} = \sqrt{2gL(1-\cos\theta)}$$

where θ is the initial angle. In computing this quantity, you may use `math.sqrt`, but ***you must use your own cos function, which you will define next.*** Specify an appropriate precision when you call `cos`.

```
def pendulum(GRAV, angle, radius) :
    """Given the force of gravity GRAV [m/s2]and the initial angle
    of a pendulum [rad] and the radius of its arc [m], return the
    period [s] and max velocity [m/s] of the pendulum."""

    period = 2.0 * math.pi * math.sqrt(radius / GRAV)

    energy = radius * (1.0 - cos(angle, 0.0001)) * GRAV
    velocity = math.sqrt(2.0 * energy)

    return period, velocity
```

Note: there is extra space on page 8

MARKS

B.4 Function cos [4 MARKS]

[4] Define a function `cos(xx,eps)` to evaluate the following infinite series for the trigonometric cosine function:

$$\cos(x) = 1 - \frac{x^2}{2!} + \frac{x^4}{4!} - \frac{x^6}{6!} + \frac{x^8}{8!} - \ldots$$

Details:

- Use the `def` statement below.
- `xx` is the angle in radians at which you will evaluate the function. Assume it is a `float`.
- `eps` is a small number—add all the terms in the series that are greater than `eps` in absolute value, and only those terms. Assume it is a positive `float`.
- Return the value of the cosine function to the calling code.
- Do not print any output from this function.
- You do not have to have a doc string and comments.

```
def cos(xx,eps) :

    count = 0
    term = 1.0
    total = 0.0
    xsq = xx * xx

    while abs(term) > eps :
        total += term
        count += 1
        term = -term * xsq / (2. * count) / (2. * count - 1.)

    return total
```

Note: there is extra space on page 8

Name

MARKS

B.5 Function `plotSwing` [5 MARKS]

[5] Define a function `plotSwing` that matches the call from `main`, to plot the path of a swing, as shown in the figure below. It must contain a title, and labels for the axes. Assume the support point for the swing is at (0,0).

There are two curves to plot:

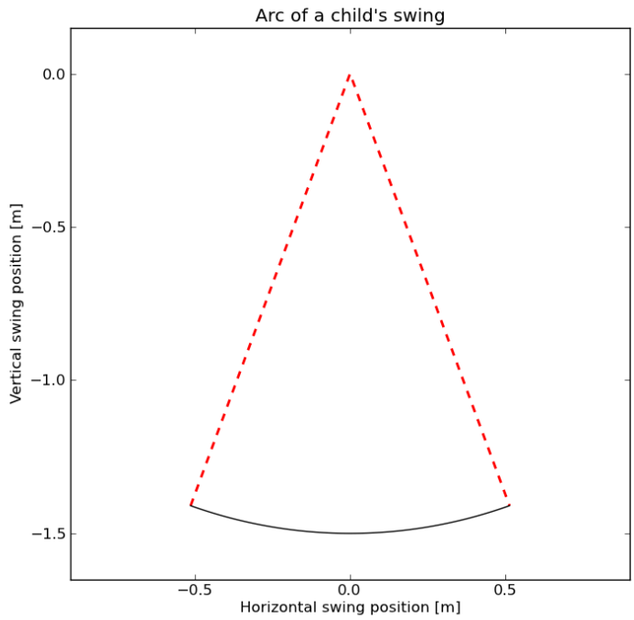
- a red dashed line from one end of the arc to the other, through the support point;
- a black continuous line from one end of the arc to the other, along the arc.

Every point on the arc itself is of the form:

$$x = L \sin \alpha$$

$$y = -L \cos \alpha$$

where α is an angle between the initial angle and the negative of the initial angle, and, as before, L is the length of the chains supporting the swing. You may use `numpy` arrays in this function, and you may use `np` functions, including `np.cos`.




```
def plotSwing(GRAV, initialAngle, radius) :
    """Plot the initial/final position of a swing, and its path."""

    # first the initial position
    xs = radius * math.sin(initialAngle) * np.array([-1., 0, 1.])
    ys = radius * cos(initialAngle, 1.e-5) * np.array([-1., 0, -1.])

    plt.plot(xs, ys, "r--")

    # now the arc
    angles = np.linspace(-initialAngle, initialAngle, 100)
    xs = radius * np.sin( angles )
    ys = -radius * np.cos( angles )
    plt.plot(xs, ys, "k-")
    plt.title("Arc of a child's swing")
    plt.xlabel("Horizontal swing position [m]")
    plt.ylabel("Vertical swing position [m]")
    plt.show()
    return
```

Note: there is extra space on page 8

MARKS

Extra space for answering any part of the programming question. Keep going; there are more questions ahead.

MARKS

Part C: 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?

```
Output: |-0.142857142857     |
```

a) `print "Output: |%-20e|" % (-1./7)`
b) `print "Output: |%20f|"  % (-1./7)`
c) `print "Output: |%-20g|" % (-1./7)`
d) `print "Output: |%-20s|" % (-1./7)`
e) `print "Output: |%20r|"  % (-1./7)`

[1] 2. Which of the following statements about `while` loops is ***not*** correct?

a) Indented instructions in a `while` loop are executed an integer number of times.
b) A `for` loop is preferred to a `while` loop when looping a known number of times.
c) If a `while` loop begins: `"while 'False' :"` it will quit looping immediately.
d) When a `while` statement is executed, the indented instructions in the loop may be executed zero times.
e) A `while` loop beginning `"while stillGoing :"` that loops once must change the value of variable `stillGoing`, or it will never quit looping.

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

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

[1] 4. After the following code has been executed, what is the value of aa?

```
def func(xx) :
    # assume xx is a list with 3 elements
    if xx[2] < xx[1] :
        if xx[2] < xx[0] :
            xx[0], xx[2] = xx[2], xx[0]
    elif xx[1] < xx[0] :
        xx[0], xx[1] = xx[1], xx[0]

aa = [5,1,3]
func(aa)
```

a) `[5,1,3]`
b) `[1,5,3]`
c) `[1,3,5]`
d) `[3,1,5]`
e) `[3,5,1]`

[1] 5. In evaluating the following expression, which operation is done ***first***?

```
-2**4 * 3 < xx + 18 or aa
```

a) `-`
b) `**`
c) `*`
d) `<`
e) `+`
f) `or`

MARKS

[1] 6. In evaluating the following expression, which operation is done ***last***?

```
-2**4 * 3 < xx + 18 or aa
```

a) -
b) **
c) *
d) <
e) +
f) or

[1] 7. Which of the following Python expressions does ***not*** correctly simulate the total after rolling two 6-sided dice?

a) `random.randint(1,7) + random.randint(1,7)`
b) `np.sum( np.random.randint(1,7,2) )`
c) `int(random.random() * 6) + int(random.random() * 6) + 2`
d) `np.random.randint(1,7) + np.random.randint(1,7)`
e) `random.randrange(1,7) + random.randrange(1.7)` `1.7 is a typo`

[1] 8. Which of the following is NOT a valid assignment given this variable definition:

```
tup = (1, "dog", abs)
lst = [2, 'cat', dir]
```

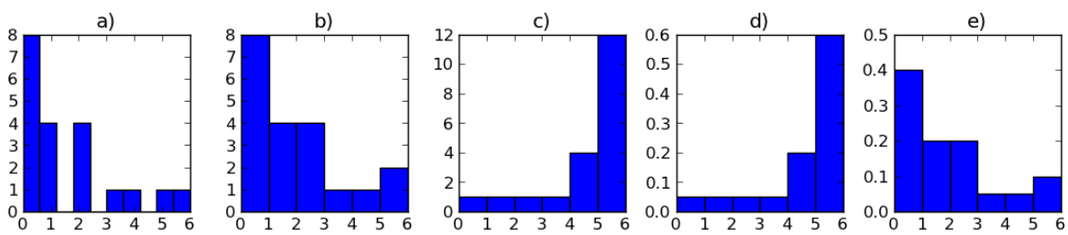
a) `tup += (4.7,)`
b) `lst += [4.7]`
c) `lst[0] += 2`
d) `tup[0] += 2`
e) `lst[1] += "ged"`

[1] 9. What is meant by Mersenne Twister?

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

[1] 10. Which of the plots below corresponds to the following instructions? `answer (e)`

```
nums = np.array([1, 2, 4, 0, 0, 6, 1, 0, 1, 0, 3, 0, 1, 0, 2, 0, 5,
                 0, 2, 2])
plt.hist(nums, bins=max(nums), normed=True)
plt.show()
```



Name

MARKS

Part D: Programming (5 MARKS)

[5] Define a function `drawSquare` that produces either a filled or empty square using a random letter as a drawing character. For example, if the size of the square is 5, then `drawSquare` would produce one of the following shapes (possibly using a different letter):

```
W W W W W          L L L L L
W       W          L L L L L
W       W          L L L L L
W       W          L L L L L
W W W W W          L L L L L
```

Details:

- The drawing character is to be a ***randomly selected upper case letter***.
- The size of the square is given by the `size` parameter. Assume without checking it is an integer greater than 0. Make sure your code works for values of 1 or 2.
- The `filled` parameter can have ***any type or value***. If `bool(filled)` is **True**, fill in the interior with the drawing letter. If not, put blanks in the interior. Note there is a blank between each pair of letters, even in the `filled` case.
- `drawSquare` does not actually print the square; instead ***it returns a string*** that when printed will look like a square.
- Complete the import line if you need one.

```
import _________________ random
def drawSquare(size, filled) :

    char = chr( int(ord('A') + 26 * random.random()) )
    filledLine = " ".join(size * [char]) + '\n'
    if size < 3 :
        square = size * filledLine
    else :
        unfilledLine = char + (2 * size - 3) * ' ' + char + '\n'
        square = filledLine + (size - 2) * unfilledLine + filledLine
    return square
```

Name

MARKS

Part E: Short Answer (5 MARKS)

[5] The function below is supposed to generate a 2-dimensional random walk with `nSteps` steps and a specified maximum step `maxStep`, starting at `start`, as in Assignment 4. It returns a tuple containing arrays of *x* positions and *y* positions. The code contains six errors that were not in the original. 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.

```
1   import numpy as np
2       def genPositions(start, nSteps, maxStep) :
        """Create a tuple of x-position array and y-position array for a 2D
        random walk starting at start, with nSteps steps, with the max
        stepsize being maxStep"""
3       stepSize = np.random.random(nSteps + 1) * maxStep
4       angle    = np.random.random(nSteps + 1) * (2.0 * np.pi)
5       xStep = maxStep * np.cos(angle)
6       xStep[0] = start[0]
7       yStep = stepSize * sin(angle)
8       yStep[0] = start[0]
        # add up the steps
9       for step in nSteps :
10          xStep[step + 1] += xStep[step]
11          yStep[step + 1] += yStep(step)
12      return (xStep, yStep)
```

(1) (line 2) Problem: the def command is lined up with the following lines; Fix: move the def command to the left margin.
(2) (line 11) Problem: wrong brackets. Fix: yStep[step]
(3) (line 5) Problem: incorrect variable—maxStep should be stepSize. Fix: change it.
(4) (line 7) Problem: missing library. Fix: use np.sin.
(5) (line 8) Problem: incorrect constant—starting y value should be start[1]; Fix: change start[0] to start[1]
(6) (line 9) Problem: missing function—nSteps isn't a collection. Fix: range(nSteps)

Extra space