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 12, 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:	_____	/ 9
B:	_____	/ 8
C1-C2:	_____	/ 7
C3-C4:	_____	/ 9
D:	_____	/ 9
E:	_____	/ 8
Total:	_____	/50

Part A: Predict the output (9 × 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(2 - 5 // 2 / 2)`	`1.0`
		`[order of operations, types]`
		`half for int 9`

2. [1]	`print(set("banana") - set("cab"))`	`{'n'}`
		`set operations`
		`half for 'n' alone`

3. [1]	`print(3 <= 3 < 3 + 3)`	`True`
		
		`[Boolean expressions]`

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

5. [1]	`print("monkey"[4:-1])`	`"e"`
		
		`[slice of a tuple, neg. index] full for no quotes`

6. [1]	`print([jj * (jj not in "aeiou") for jj in 'banana'])`	`['b', '', 'n', '', 'n', '']`
		
		`[list comprehension, repeat]`
		`half for ['b', 'n', 'n']`

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

8. [1]	`DL='\N{DEGREE SIGN}'` `print(3*DL)`	`°°°` `String processing; half if they don't remember sign`
9. [1]	`print(2 + (2j)**2)`	`-2+0j` `[complex number; accept -2. + 0.j or any combo; -2 or -2. gets 0.5 -- not complex]`

MARKS

Part B: Programming – [5 + 3 MARKS]

B.1 Check New Password [5 MARKS]

[5] Define a function `newPassword` that asks a user to enter a new password, and checks it against a set of rules. If the password satisfies all the rules, return it as the value of the function. If the password doesn't satisfy some of the rules, print out a warning message and ask the user for another password. Print out a warning for each rule that is not satisfied. Ask a ***maximum of three times***, after which the function should return the value `None` if the password is still invalid. Rules:

- The length of the password must lie between the values of the two parameters, `minLen` and `maxLen`. It can equal either of them. Use an assert statement to ensure that `minLen` is at least 3 and `maxLen` is not smaller than `minLen`.
- It must contain an uppercase letter, a lowercase letter and a digit. Hint: ***call the function you will define on the next page***

```
def newPassword(minLen, maxLen) :
    """Ask the user to provide a new password, and check it against
    several rules. If it passes, return the password. Give the user
    3 chances, and return None if the password is still invalid.
    Length must lie between minLen and maxLen inclusive."""

    assert 3 <= minLen <= maxLen, (
        "Bad lower bound %s or upper bound %s" % (minLen, maxLen))
    prompt = "Enter a new password: "
    count = 3
    warn = "\b"

    while warn and count :
        count -= 1
        pw = input(warn + '\n' + prompt)
        warn = ''
        if not (minLen <= len(pw) <= maxLen) :
            warn += ("Password %s:length not between %d and %d\n"
                     % (pw, minLen, maxLen))
        if not howMany(pw, 'A', 'Z') > 0 :
            warn += ("Password %s: no uppercase letter\n" % pw)
        if not howMany(pw, 'a', 'z') :
            warn += ("Password %s: no lowercase letter\n" % pw)
        if not howMany(pw, '0', '9') :
            warn += ("Password %s: no digit\n" % pw)
    if warn :
        pw = None
    return pw
```

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

A. [1] assert
B. [1] while loop with count
C. [1] input with prompt and warn
D. [1] tests with warning
E. [1] return value

Name

MARKS

B.2 Function howMany [3 MARKS]

[3] Write a function definition for a function `howMany` that returns a count of how many characters in `word` lie in a character range. The characters to be checked for are in consecutive positions in the Unicode character set. The initial character is given by parameter `startChar`, and the final character is `endChar`. The count includes both those characters. For example, a call `howMany("Zoo",'A','Z')` returns 1 because there is one uppercase letter between `'A'` and `'Z'` in `'Zoo'`.

```
def howMany(word, startChar, endChar) :
    """Return the count of characters in word that lie between
    startChar and endChar inclusive. For instance,
    howMany('1X23','0','9') returns 3."""

    inList = [startChar <= ch <= endChar for ch in word]
    return sum(inList)
```

A. [2] check which letters are in range
B. [1] return a count

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

C. This section defines one program; fill in parts C.1 to C.4 on later pages

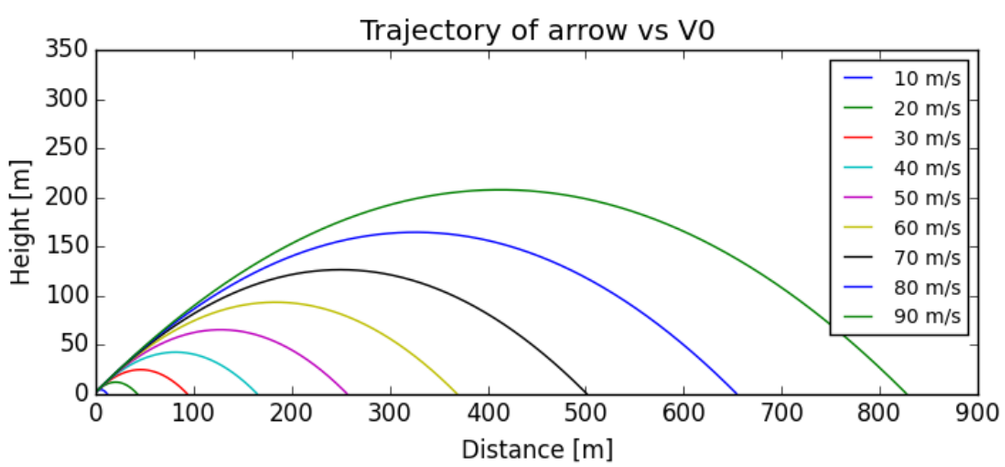
This program determines the effect of initial velocity on the flight of an arrow shot upward at a 45° angle from a height of 2 m. The main function is done for you below. Sample table and graphical output are shown. Reproduce them as closely as possible.

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

#.............................................................
global constants
g_ = 9.81 # [m/s^2] acceleration due to gravity at earth's surface
NUM_POINTS = 10001 # number of points to use for times and distances

#.............................................................main
def main() :
    """Simulate an arrow's flight"""
    # ARROW PROPERTIES
    y0 = 2.0 # [m] initial height
    x0 = 0.0 # [m] initial horizontal position
    angle0 = 45 # [degrees] angle from horizontal
    compareVelocities(angle0, x0, y0)
    return

def calcSpeeds(times, xs, ys) :          # C.1 see later page

def trajectory(angleDeg,veloc,x0,y0) :  # C.2 see later page

def compareVelocities(angle0, x0, y0) : # C.3 see later page

def quadSolve(coefs) :                   # C.4 see later page

# Start the program!
main()
```

Sample output

V0 [m/s]	Time [s]	Distance [m]	Height [m]	Min V [m/s]
10	1.7	11.9	4.5	7.1
20	3.0	42.7	12.2	14.1
30	4.4	93.7	24.9	21.2
40	5.8	165.1	42.8	28.3
50	7.3	256.8	65.7	35.4
60	8.7	369.0	93.7	42.4
70	10.1	501.5	126.9	49.5
80	11.6	654.4	165.1	56.6
90	13.0	827.7	208.4	63.6

Sample plot



MARKS

C.1 to C.4 together make up a program; you may want to look them all over.

C.1 Function `calcSpeeds` [3 MARKS]

[3] Define a function `calcSpeeds` that determines average speeds in small intervals of time.

Parameters (may be lists or arrays):

- `times`: [s] a sequence of increasing times. Assume there are at least 2 times.
- `xs`: [m] horizontal positions of an object at the given times (same length as `times`)
- `ys`: [m] vertical positions of an object at the given times (same length as `times`)

Returns: a numpy array containing estimates of the speed of the object in each interval between consecutive times. Formula for the *i*'th speed:

$$v_i = \frac{\sqrt{(x_{i+1} - x_i)^2 + (y_{i+1} - y_i)^2}}{t_{i+1} - t_i}$$

Use numpy array calculations, and do NOT code any loops.

```
def calcSpeeds(times, xs, ys) :
    """Estimate the average velocity in each consecutive
    time interval between values in times. xs and ys are the
    x and y positions at each time. Parameters may be
    lists or arrays. Their lengths are equal. The return
    value is an array of velocities.
    """
    # start coding on next line
    times = np.array(times)
    xs = np.array(xs)
    ys = np.array(ys)
    lengths = np.sqrt((xs[1:] - xs[:-1])**2
                      + (ys[1:] - ys[:-1])**2)
    return lengths / (times[1:] - times[:-1])

A: [1] Convert to arrays
B: [1] Calculate lengths
C: [1] Calculate and return speeds
```

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

MARKS

C.1 to C.4 together make up a program; you may want to look them all over.

C.2 Function `trajectory` [4 MARKS]

[4] The x and y positions of an arrow at time t after release are given by these expressions:

$$\left.\begin{array}{rl} x = & x_0 + V_x \cdot t \\ y = & y_0 + V_y \cdot t - \frac{1}{2}gt^2 \end{array}\right\} \text{ where } \left\{\begin{array}{rl} V_x = & V\cos\theta \\ V_y = & V\sin\theta \end{array}\right.$$

This function evaluates the horizontal and vertical components of velocity and uses them to determine x and y positions of an arrow released at $t = 0$. This function sets up a coefficient list and calls `quadSolve` to find out when to terminate the simulation because the arrow has hit the ground (that is, when $y = 0$). It divides the time between release and landing into `NUM_POINTS-1` equal intervals and determines `NUM_POINTS` times. It evaluates the x and y values at each of these times (`g_` is a global constant). It uses numpy arrays (numpy is already imported) to perform the computations, so there is no loop. Fill in the blanks to achieve these objectives using good programming practices.

```
def trajectory(angleTheta,velocV,x0,y0) :
    """Simulate the flight of an arrow shot into the air at an angle
    angleTheta in degrees with initial velocity velocV, starting at
    position (x0,y0). Use NUM_POINTS times between the start at time
    0 and its return to earth. Return the times, x positions and
    y positions as three arrays."""

    # ARROW PROPERTIES
    angleRad = ____angleTheta * np.pi / 180____________________________

    velX = veloc * np.cos(angleRad)

    velY = veloc * np.sin(angleRad)

    yCoefs = ____(-g_/2, velY, y0)_____________________________________

    timeToLand = quadSolve(yCoefs)[1] # larger root

    # calculate evenly spaced times

    times = ____np.linspace(0,timeToLand,NUM_POINTS)___________________

    xs = x0 + times * velX

    # In the next line, do NOT call evalPoly; it is not available.

    ys = ____(yCoefs[0] * times + yCoefs[1]) * times + yCoefs[2]_______

    return times, xs, ys
```

A: [1] Initialize angle (rads), velocities
B: [1] set up coefs and call quadSolve
C: [1] generate times, and return times, xs, ys
D: [1] generate x and y values

C.1 to C.4 together make up a program; you may want to look them all over.

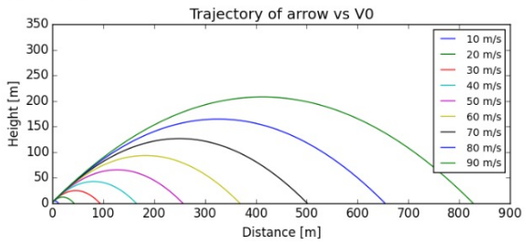
C.3 Function `compareVelocities` [5 MARKS]

[5] Compute the trajectory of an arrow for each of several initial velocities. Display the results in a table, one row per initial velocity, and in a plot, one curve per initial velocity. Use the initial velocities shown as V0 in the sample output. Call `trajectory` and `calcSpeeds`. The Distance and Height columns are maximum *x* and *y* values. The Min V column is the minimum total velocity observed in the values returned from `calcSpeeds`.

Sample output

V0 [m/s]	Time [s]	Distance [m]	Height [m]	Min V [m/s]
10	1.7	11.9	4.5	7.1
20	3.0	42.7	12.2	14.1
30	4.4	93.7	24.9	21.2
40	5.8	165.1	42.8	28.3
50	7.3	256.8	65.7	35.4
60	8.7	369.0	93.7	42.4
70	10.1	501.5	126.9	49.5
80	11.6	654.4	165.1	56.6
90	13.0	827.7	208.4	63.6

Sample plot




```
def compareVelocities(angle0, x0, y0) :
    """Evaluate, and display in table and plot, arrow flight
    properties for different initial velocities from 10 to
    100 m/s. No return value."""
    print("V0 [m/s]   Time [s]  Distance [m]  Height [m]  Min V [m/s]")
    fig = plt.figure()
    fig.clf()
    fig.add_subplot(111,aspect="equal")
    plt.title("Trajectory of arrow vs v0")
    plt.xlabel("Distance [m]")
    plt.ylabel("Height [m]")
    # Enter the rest of the function definition below.

    for v0 in range(10,100,10) :
        times, xs, ys = trajectory(angle0, v0, x0, y0)
        vels = calcSpeeds(times, xs, ys)
        timeToLand = times[-1]
        plt.plot(xs,ys,label=("%d m/s" % v0))
        print("%5d %11.1f %11.1f %11.1f %13.1f"
            % (v0, timeToLand, max(xs), max(ys), min(vels)))

    plt.ylim(0,350)
    plt.legend(loc="best", fontsize=10)
    plt.show()
    return

A: loop over velocities
B: generate times, xs, ys, velocities
C: print row of table (min V, max xs and ys)
D: plot y vs x
E: plot legend and show plot
```

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

MARKS

C.1 to C.4 together make up a program; you may want to look them all over.

C.4 Function quadSolve [4 MARKS]

[4] The code below returns a tuple of the two roots of a quadratic equation with given coefficients. The coefficients are in a list in the order $[a, b, c]$ for the quadratic equation $ax^2 + bx + c = 0$. The return values are floats for real roots, and complex numbers for complex roots. The solution formula is ...

$$x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$

The code below was originally correct, but 5 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 four more.

```
    import numpy as np
1   def quadSolve(coefs)
2       aa; bb; cc = coefs # first coefficient is quadratic, last is constant
3       assert aa != 0, "Invalid quadratic coefficient 0 for a quadratic equation"
4       discriminant = bb**2 - 4 * aa * cc # tells whether roots are real
5       if discriminant < 1.e-14 :         # tiny; effectively equal roots
6           roots = 2 * [-bb / 2 * aa]     # duplicate roots
7       elif discriminant < 0 :            # complex roots
8           x0 = (-bb + np.sqrt(discriminant + 0j)) / (2 * aa)
9           roots = [x0, x0.conjugate()]   # changes sign of imaginary part
10      else :
11          sqDisc = np.sqrt(discriminant) # real square root of discriminant
12          roots = [(-bb + sqDisc) / (2 * aa), (-bb - sqDisc) / (2 * aa)]
13          bigAbs = roots[abs(roots[1]) < abs(roots[0])] # bigger in abs value
14          smallAbs = cc / aa / bigAbs    # more accurate estimate of other root
15          roots = sorted([smallAbs,bigAbs]) # roots sorted by value
16      return tuple(roots)
```

(1) line 1... ':' is missing in the **def** statement; add it after the closing parenthesis ')'
(2) line 2: tuple assignment requires comma separation, not semi-colon; should be aa, bb, cc.
(3) line 5: need abs on discriminant
(4) line 6: denominator should be in brackets: (2 * aa)
(5) line 13: < should be >

Name

MARKS

D.1 Function sinPrecise [5 MARKS]

[5] Recall the terms of the sine series: $\sin(x) = \frac{x}{1} - \frac{x^3}{3!} + \frac{x^5}{5!} - \frac{x^7}{7!} + \ldots$

In Python, floating-point arithmetic has about 16 digits of precision, whereas integer arithmetic has unlimited precision. We can calculate the result of this series to (for example) 40 digits of precision by multiplying the terms by a big integer (10^{40}) and using integer arithmetic:

$$10^{40}\sin(x) = \frac{10^{40}x}{1} - \frac{(10^{40}x)^3}{3!\,(10^{40})^2} + \frac{(10^{40}x)^5}{5!\,(10^{40})^4} - \frac{(10^{40}x)^7}{7!\,(10^{40})^6} + \ldots$$

Write one function `sinPrecise` with two parameters: `xx` (the angle in radians; a float) and `prec` (the number of digits of precision; an int) that returns sin(xx). Details:

- Do ***not*** call ***any*** library functions. Evaluate the function by summing the series.
- Re-use the previous term to evaluate the next term.
- Use only integer operations when evaluating the next term.
- Calculate and use terms with magnitudes greater than zero.

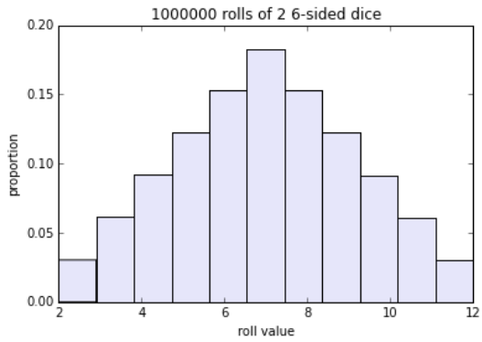
You may call the function `angleBig(angle, digits)` that returns the integer result `10**digits * angle`. YOU DO NOT HAVE TO WRITE THIS FUNCTION.

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

```
def angleBig(angle, digits) :
   """Return 10**digits * angle as an int"""

def sinPrecise(xx, prec) :
     """Evaluate sin(xx) with prec digits of precision."""
     BIG10_POWER = 10**prec
     BIG10_SQ = BIG10_POWER**2
     bigX = angleBig(xx, prec)
     bigXsq = bigX * bigX
     count = 0
     total = 0
     term = bigX
     while abs(term) > 0 :
           count += 1
           total += term
           term = (-term * bigXsq // (2*count)
                 // (2*count + 1) // BIG10_SQ
     return total / BIG10_POWER

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

MARKS

D.2 Function diceRoll (4 marks)



[4] This function simulates a series of random rolls of fair dice and plots a histogram of the results. It uses `numpy` vector operations to perform the random rolls.

Write one function `diceRoll` with parameters `numDice` and `numRolls`. Details:

- Define a global constant for the number of sides on one die, and set it to 6.
- You are permitted only one loop (for the number of dice).
- Use an array of `numRolls` integers to contain the results of the rolls.
- Plot and show a histogram of the results of the rolls:
- Plot the results so that the area of the columns sums to 1 (use `normed=True`).
- The number of bins is equal to the number of possible results of a single roll. For example, with 2 six-sided dice, the number of possible results is 11 (2…12).
- Provide meaningful x- and y-axis labels, and a plot title (see the sample plot).
- Do not code any input or other output in this function.

```
import numpy as np

# Define your global constant here

SIDES = 6

def diceRoll(numDice, numRolls) :
    """Simulate random rolls of dice and plot a
    histogram"""
    rolls = 0 * np.arange(numRolls)

    for roll in range(numDice):

        rolls += np.random.randint(1, SIDES+1, numRolls)

    bins = numDice * (SIDES - 1) + 1

    plt.title("%d rolls of %d %d-sided dice" %
      (numRolls, numDice, SIDES))

    plt.ylabel("proportion")

    plt.xlabel("roll value")

    plt.hist(rolls, bins, normed=True)

    plt.show()

    return
```

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

```
A: [2 marks] Prepare the data: sum of rolls for each case
B: [2 marks] Produce the plot.
```

MARKS

Part E: Multiple Choice + Expression [8 x 1 MARK]

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

[1] 1. Given `phrase = ' sunny ways, my friends'`, which of the following will produce the string `' Sunny Ways, My Friends'`?

a)
```
''.join([[char, char.upper()][phrase[pos-1] == ' ']
        for pos, char in  enumerate(phrase)])
```
b)
```
''.join([[char.upper(), char][phrase[pos-1] == ' ']
        for pos, char in enumerate(phrase)])
```
c)
```
''.join([[char, char.upper()][phrase[pos-1] != ' ']
        for pos, char in enumerate(phrase)])
```
d)
```
''.join([char.upper() for pos, char in enumerate(phrase)])
```
e)
```
''.join([[char.lower(), char][phrase[pos-1] == ' ']
        for pos, char in enumerate(phrase)])
```

[1] 2. Given `word="liberated"`, which of the following produces the result `"bad"`?

a) `word[2:3]`
b) `word[0:3]`
c) `word[2:3:-1]`
d) `word[2::3]`
e) `word[0::3]`

[1] 3. Given `nn = -123`, which of the following print statements results in the given output?

```
|-0123|
```

a) `print("|%05d|" % nn)`
b) `print("|%5d|"  % nn)`
c) `print("|%-5d|" % nn)`
d) `print("|%5s|"  % nn)`
e) `print("|%-5g|" % nn)`

[1] 4. If `a` and `b` have int values, which of the following will make `nums` refer to a collection of two pseudo-random integers in the range `[a, b)` (that is, including `a` but excluding `b`)?

a) `nums = np.random.random(2)`
b) `nums = np.random.randint(a,b,2)`
c) `nums = [random.random() for item in range(2)]`
d)
```
nums = []
for xx in range(2):
    nums += int(random.random()*(b-a+1)) + a
```
e) `nums = np.array([np.random.random(a), np.random.random(b)])`

[1] 5. Given the statements below, which expression will NOT assign an array to `result`?

```
import numpy as np
arr = np.arange(10)
```

a) `result = np.cos(arr)`
b) `result = np.repeat(np.sum(arr), len(arr))`
c) `result = np.cos(arr) + np.sum(arr)`
d) `result = np.sum(np.cos(arr)) + arr`
e) `result = np.max(arr) + np.min(arr)`

MARKS

[1] 6. 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

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

```
import numpy as np
var1 = np.linspace(1,2,3)
var2 = var1[1:4]
var2[-2] = -2
print(var1)
```

a) [1 -2 2]
b) [1. 1.5 2.]
c) [1. -2. 2.]
d) [1. 2. 3.]
e) [1. -2. 3.]

[1] 8. 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:

$$\left(\lfloor x \rfloor - \frac{b}{\frac{-2}{4}} \right) \cdot \sin(3)$$

Put expression here

```
short form: (math.floor(xx) + 2. * bb) * math.sin(3.)
long form:  (math.floor(xx) - (bb / (-2. / 4.))) * math.sin(3.)
math
```

MARKS

Extra space

Extra space