THE UNIVERSITY OF MANITOBA

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

Name
Student number

Section (please check one):

☐ A01 (Andres EITC E2-110)

☐ A02 (Melvin Buller 207)

December 12, 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:	_______/25
C:	_______/10
D:	_______/ 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 2**3 // 5 % 4`	`1`
		`[order of operations]`

2. [1]	`print tuple(np.linspace(1,2,2))`	`(1.0, 2.0)`
		`[combined array operations, cast]`

3. [1]	`print 1 <= 1 >= 2**1`	`False`

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

5. [1]	`print 'alphabet'[-5:5]`	`‘ha’`
		`[slice of a string, neg. index]`

6. [1]	`print ('Jan', 'Feb', 'Mar', 'Apr', 'May', 'Jun')[-5:2]`	`(‘Feb’,)`
		`[slice of a tuple]`

7. [1]	`print [jj + 5 // 3 for jj in [0, 5, 3, 3] ]`	`[1,6,4,4]`
		`[list comprehension]`

8. [1]	`print (array([3, 3, 0, 4]) * 2 - 2)`	`[4  4  -2  6]`
		`[array processing]`

9. [1]	`print "123" + "877"`	`"123877”`
		`[string concatenation]`

10. [1]	`print 'The quick brown fox'.split()`	`['The', 'quick', 'brown', 'fox']`
		`[string function]`

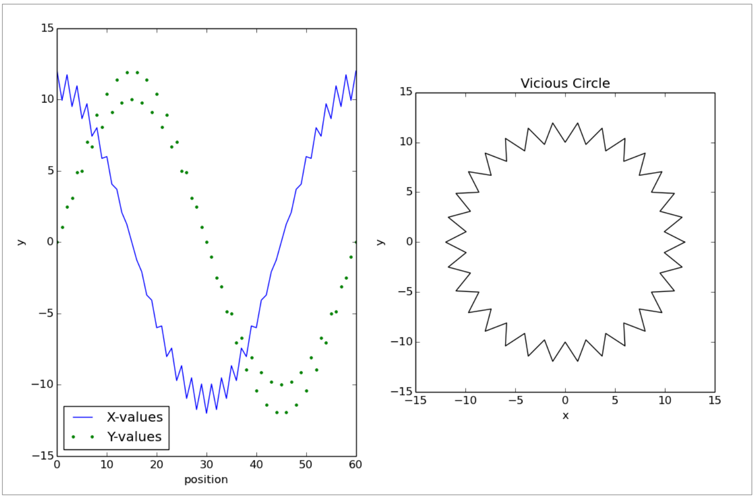
Name

MARKS

Part B: Programming – Full Program (4 + 2 + 7 + 4 + 6 + 2 = 25 MARKS)

A 'vicious circle' is a shape where half the points lie on an inner circle and half lie on an outer circle. Write a program that reads input about a vicious circle, then plots it, and finds its area. Below is the printed output from one run of the program, and the plotted output is shown below that on the left.

```
Analyzing a Vicious Circle

Enter the number of teeth in a vicious circle: 30

Enter the ratio of outer to inner radius (between 1 and 2): 0.8

Out of range; enter a value between 1 and 2.
Enter the ratio of outer to inner radius (1 to 2): 1.2

Area with inner & outer radii 10 & 12 [cm] and 61 points is 376 [cm^2]

Program terminating
```



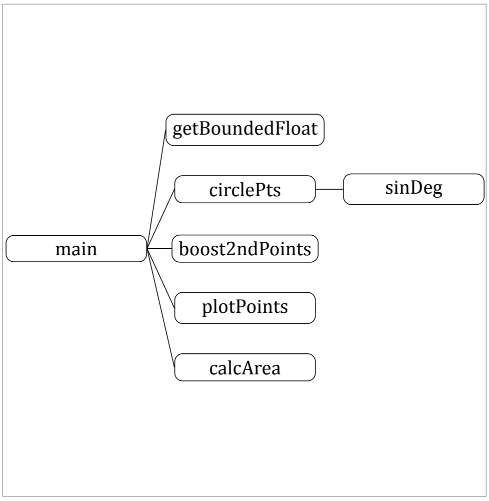


Figure 1 : Left: Program Plot Output; Right: Structure Chart

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. You will write the five underlined functions in `main` (no choice), and also `sinDeg`, used at a lower level. (See the structure chart above right.) Each function will be marked separately—even if you have problems with one, you can get full marks on others.

```
import matplotlib.pyplot as plt
import numpy as np
DEGREES_IN_CIRCLE = 360.
def main() :
    print "Analyzing a Vicious Circle"
    numTeeth = int(getBoundedFloat(3, 100,
            "Enter the number of teeth in a vicious circle: "))
    numPoints = 2 * numTeeth + 1
    ratio = getBoundedFloat(1.0, 2.0,
          "Enter the ratio of outer to inner radius "
          "(between 1 and 2): ")
    radius = 10.
    points0 = circlePts(radius, numPoints)
    points1 = boost2ndPoints(points0, ratio)
    plotPoints(points1, "Vicious Circle")
    area = calcArea(points1)
    print ("\nArea with inner & outer radii %g & %g [cm] "
           "and %d points is %g [cm^2]"
           % (radius, radius * ratio, numPoints, round(area)) )
    print "\nProgram terminating"
    return
```

Do NOT put any answers on this page.

Name

MARKS

B.1 Function `getBoundedFloat` [4 MARKS]

[4] Define a function `getBoundedFloat(loBnd, hiBnd, prompt)` that matches the call from `main`. The parameters `loBnd` and `hiBnd` are lower and upper bounds on the value the user can enter. The user can enter either of these values or anything in between. The parameter `prompt` is the string that this function shows the user to get input. The input is a single float value. 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 non-float input. You just have to check that the input's value lies in the correct interval. If the user enters an unacceptable value, print a warning message before prompting again. The warning message must contain `loBnd` and `hiBnd`, formatted. For example, entry of the ratio with a user mistake might look like this:

```
Enter the ratio of outer to inner radius (1 to 2): 0.8

Out of range; enter a value between 1 and 2.
Enter the ratio of outer to inner radius (between 1 and 2): 1.2
```

```
def getBoundedFloat(loBnd, hiBnd, prompt) :
    """Ask the user for a float between loBnd and hiBnd using prompt,
    and keep asking till you get a valid float. Then return it."""

    warning = "\n"
    while warning :
        usrIn = raw_input(warning + prompt)
        usrVal = float(usrIn)
        warning = ''
        if not (loBnd <= usrVal <= hiBnd) :
            warning = (
                "Out of range; enter a value between %g and %g.\n"
                % (loBnd, hiBnd) )

    return usrVal

A. [1] input, convert to float, return
B. [2] loop executes once, repeats on invalid input only
C. [1] test and warning message
```

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

Note: there is extra space on page 8

Name

MARKS

B.2 Function `circlePts` [2 MARKS]

[2] Define a function `circlePts(radius, numPoints)` that matches the call from `main`.

- It returns a 2-element tuple containing a `numpy` array with `numPoints` *x* points and a `numpy` array of `numPoints` *y* points. Pairs of (*x*,*y*) points from these two arrays are spaced evenly along the circumference of a circle with the given `radius`, centred at (0,0). Points at ***both 0° and 360° are included.***
- The formulas for *x* and *y* are:

$$x = r\cos\theta_j$$
$$y = r\sin\theta_j$$

  where θ_j is one of the equally spaced angles from 0° to 360°.
- Call the functions `sinDeg` and `cosDeg` to evaluate sin and cos functions (***not*** `np` or `math` functions). These functions take ***array arguments*** in ***degrees***, not radians. You will program `sinDeg` in B.3. Assume `cosDeg` is also available.
- ***Note: in this and the following functions, you must accept and/or return numpy arrays, but you may use array calculations or not, as you please.***

```
def circlePts(radius, numPoints) :
    """Return arrays of x values and y values from numPoints points
    evenly spaced from 0 to 360 degrees along the circumference of a
    circle with given radius, centred at the origin (0,0)."""


    angles = np.linspace(0., 360., numPoints)
    return (radius * cosDeg(angles), radius * sinDeg(angles))

A. [1] Generate angles around circle
B. [1] Calculate, return x and y
```

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

Note: there is extra space on page 8

Name

MARKS

B.3 Function `sinDeg` [7 MARKS]

[7] Define a function `sinDeg` that matches the call from `circlePts`. It receives a single parameter: `degrees`, which is a ***numpy array of angles*** in units of degrees. It converts these angles to radians, then evaluates the sine function for each angle using a power series. It returns the array of sine values corresponding to the incoming `degrees`.

The following formula transforms an incoming angle x_{degrees} in degrees into a 'reduced' angle in radians between -π and π, that has the same sine value as the incoming angle. The expression "*a* mod 360" finds the ***float*** remainder when dividing *a* by 360.

$$x_{\text{rad}} = ((x_{\text{degrees}} + 180) \bmod 360 - 180)\,\frac{\pi}{180}$$

Evaluate the following series for sin(x):

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

Use exactly 15 terms of the series (that is, terms up to $x^{29}/(29!)$). (Because every reduced angle x_{rad} lies between –π and π, the series converges with 15 terms for any incoming angle.) ***Do not use any factorial function.***

```
def sinDeg(degrees) :
    """degrees is an array of angles in degrees. Return an array
    of sine values, one for each angle."""

    deg2 = (degrees + 180) % 360 - 180
    rads = deg2 * (np.pi / 180.)
    radsq = rads * rads
    count = 0
    total = 0. * rads     # to get right number of zeros
    term = rads.copy()    # copy protects rads from change
    while count < 15 :
        total += term
        count += 1
        term = -term * radsq / ((2. * count) * (2. * count + 1.))
    return total
```

```
A. [2] Convert all angles to radians
B. [1] loop exactly 15 times
C. [2] count and total, update and return, for all angles
D. [2] calculate term for all angles
```

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

Note: there is extra space on page 8

Name

MARKS

B.4 Function `boost2ndPoints` [4 MARKS]

[4] Define a function `boost2ndPoints(pointsIn, factor)` that returns a new tuple similar to `pointsIn`, except that every second point has been multiplied by `factor`.

- `pointsIn` is a 2-element tuple (`xs`, `ys`) containing two `numpy` arrays of the same length; corresponding values from `xs` and `ys` are assumed to make (x,y) points.
- Use an `assert` to check that the two arrays in `pointsIn` are the same length.
- Use the following formula to calculate the x and y points to return in a tuple, where a represents the incoming `factor`; ***every 2nd point*** (x,y) in `pointsIn` is boosted:

$$(x'_j, y'_j) = \begin{cases} (x_j, y_j) & \text{if } j \text{ is odd} \\ (ax_j, ay_j) & \text{if } j \text{ is even} \end{cases}$$

- ***Make sure you do <u>not</u> change values in*** `pointsIn`.

```
def boost2ndPoints(pointsIn,factor) :
    """pointsIn should be a tuple with two arrays, one of x-values
    and one of y-values, of the same length. factor is a number.
    Multiply every second point by factor, and return the modified
    collection of points, without changing pointsIn."""
    xs, ys = pointsIn
    assert len(xs) == len(ys), "Need same number of x and y values"
    xsOut, ysOut = np.array(xs), np.array(ys) # copies
    xsOut[0: :2] *= factor
    ysOut[0: :2] *= factor
    return xsOut, ysOut

A. [1] assert
B. [2] update xsOut and ysOut without changing pointsIn
C. [1] return values
```

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

Note: there is extra space on page 8

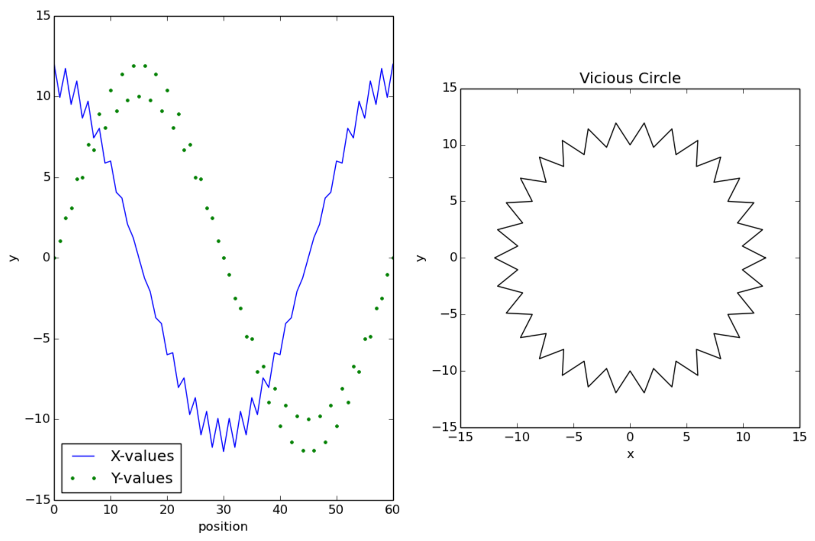
Name

MARKS

B.5 Function `plotPoints` [6 MARKS]

[6] Define a function `plotPoints(points, title)` that matches the call from `main`, to plot 2 views of a set of points in 2 subplots of a single figure, as shown below.

- `points` is a 2-element tuple (`xs`, `ys`) containing two `numpy` arrays of the same length; corresponding values from `xs` and `ys` are assuming to make (*x*,*y*) points.
- Draw a figure with two subplots, side-by-side. They must both have axis labels.
- The left subplot shows *x* and *y* values plotted separately against point number. One line must use a solid curve, and the other shows only points. A legend is required.
- The right subplot shows *y* values plotted against *x* values. It uses `title` as a title.



```
def plotPoints(points, title) :
    """points should be a tuple with two arrays, one of x-values
    and one of y-values, of the same length. Prepare two side-by-
    side subplots, one showing the x- and y-values separately,
    and one of y-values plotted against x-values."""

    fig = plt.figure()
    fig.add_subplot(121)
    xs, ys = points
    plt.plot(np.arange(len(xs)), xs, 'b-', label="X-values")
    plt.plot(np.arange(len(ys)), ys, 'g.', label="Y-values")
    plt.legend(loc="lower left")
    plt.xlabel('position')
    plt.ylabel('y')

    fig.add_subplot(122, aspect="equal")
    plt.plot(xs, ys, 'k')
    plt.title(title)
    plt.xlabel('x')
    plt.ylabel('y')
    plt.show()
```

```
A. [1] plt.figure and fig.add_subplot usage
B. [3] left subplot: 2 curves, solid and points, legend, axes
C. [2] right subplot: title, axes, curve, plt.show
```

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

Name

MARKS

Extra space for answering any part of the programming question. There is also room on the last page. Keep going; there are more questions ahead.

Name

MARKS

B.6 Function `calcArea` [2 MARKS]

[2] Define a function `calcArea(points)` that matches the call from `main`, to calculate the area defined by the provided `points`.

- `points` is a 2-element tuple (`xs`, `ys`) containing two `numpy` arrays of the same length; corresponding values from `xs` and `ys` are assuming to make (*x*,*y*) points.
- the area defined by two consecutive points (x_i, y_i) and (x_{i+1}, y_{i+1}) is the trapezoidal area:

$$A_i = \frac{1}{2}(x_{i+1} - x_i)(y_{i+1} + y_i)$$

- do ***not*** try to join the last point back to the first; if `points` defines a closed curve, the last point will already be identical to the first.

```
def calcArea(points) :
    """points should be a tuple with two arrays, one of x-values and
    one of y-values, of the same length. Calculate its area using
    trapezoidal integration and return it."""

    xs, ys = points
    total = (xs[0] - xs[-1]) * (ys[0] + ys[-1])
    total += np.sum( (xs[1:] - xs[:-1]) * (ys[1:] + ys[:-1]) )

    return abs(total) / 2.

A. [1] Calculate area of each interval
B. [1] Calculate total area and return it
```

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

Name

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: |-0.2857              |
```

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: |%20r|"    % (-2./7)`

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

[✓ a) If a `while` loop begins as follows, it will quit looping immediately:
`while 'False' :`]
b) When a `while` statement is executed, the indented instructions in the loop may be executed zero times.
c) Suppose statements in a given `while` loop are executed at least once. If the loop begins as follows, code in the loop must change the value of variable `stillGoing`, or the loop will never stop.
`while stillGoing :`
d) Indented instructions in a `while` loop are executed an integer number of times.
e) A `for` loop is preferred to a `while` loop when looping a known number of times.

[1] 3. Which of the following could be the result of the statement
`print np.random.randint(-2,6,4)` ?

a) [3 -1 2 2 0]
b) [6 -1 2 0]
c) [3 -1 2 -2 1 2]
[✓ d) [5 -1 2 2]]
e) [4 -1 2 0 2 3]

[1] 4. After the following code executes what is the value of yVals?

```
def evalPoly(poly,xVals) :
    xVals = np.array(xVals)
    fxVals = np.zeros(len(xVals))
    for coef in reversed(poly):
        fxVals = fxVals * xVals + coef
    return fxVals

coef  = [2,2,2]
xVals = np.array([1,2,3])
yVals = evalPoly(coef,xVals)
```

a) `array([  2.,   11.,   12.])`
b) `array([  6,     14,   26 ])`
c) `array([  2.,    2.,    2.])`
d) [✓ `array([  6.,   14.,   26.])`]
e) `array([  17,    17,   17])`

Name

MARKS

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

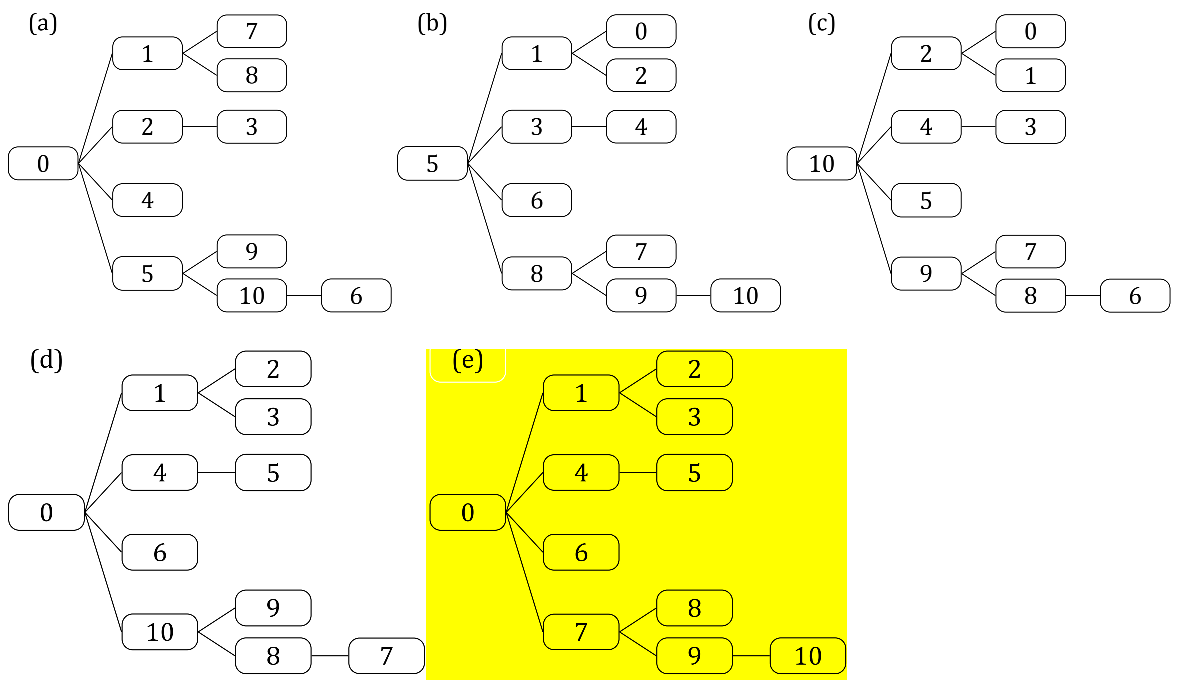
a) Lists and tuples are immutable, but strings and `numpy` arrays can change.
b) Lists and `numpy` arrays are immutable, but tuples and strings can change.
[✓ c) Tuples and strings are immutable, but lists and `numpy` arrays can change.]
d) Strings and lists are immutable, but tuples and `numpy` arrays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

[1] 6. After the following sequence of instructions is executed, what is the value of `letters`?

```
letters = "ABCDEFG"
letters = letters[-1: :-1]
letters = letters[:3][-1: :-1] + letters[3:][-1: :-1]
```

a) `"ABCDEFG"`
b) `"CBAGFED"`
c) `"EFGABCDEFG"`
[✓ d) `"EFGABCD"`]
e) `""`

[1] 7. If the numbers in the following structure charts represent the order in which you implement the corresponding functions (e.g., you always implement function 5 before function 6), which of these charts represents top-down programming?


(a) 0 → 1 (7, 8), 2 (3), 4, 5 (9, 10 (6))
(b) 5 → 1 (0, 2), 3 (4), 6, 8 (7, 9 (10))
(c) 10 → 2 (0, 1), 4 (3), 5, 9 (7, 8 (6))
(d) 0 → 1 (2, 3), 4 (5), 6, 10 (9, 8 (7))
(e) 0 → 1 (2, 3), 4 (5), 6, 7 (8, 9 (10))

[1] 8. What is meant by Horner's Method?

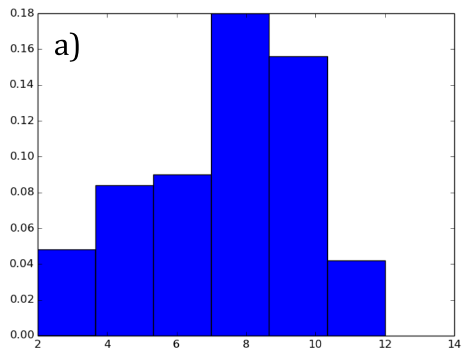
[✓ a) A method for efficiently evaluating a polynomial.]
b) A random number generator used by Python.
c) A way of determining all prime numbers up to a limit.
d) A rule describing the exponential growth of computer chip capacity.
e) The best way of summing up an infinite series.

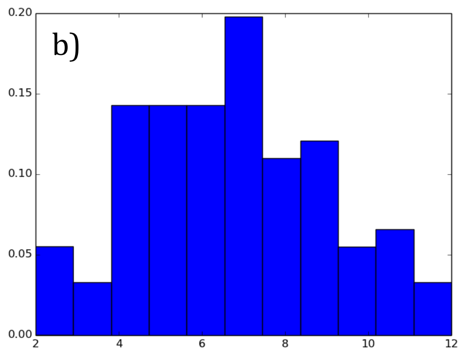
Name

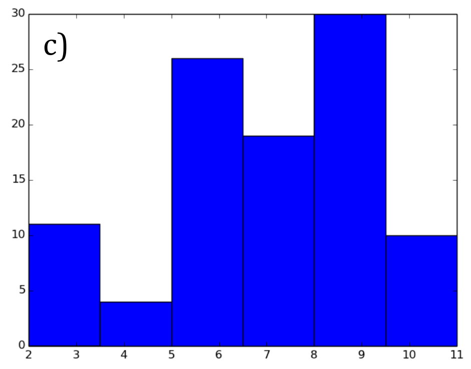
MARKS

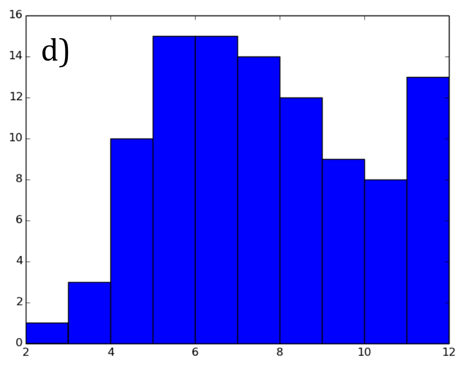
[1] 9. Which plot could result from the following code?

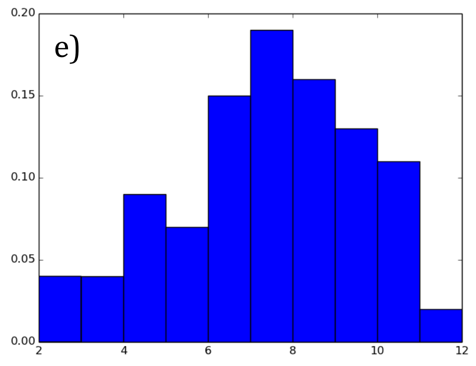
```
import numpy as np
import matplotlib.pyplot as plt
xx1 = np.random.randint(1,7,100)
xx2 = np.random.randint(1,7,100)
dice = xx1 + xx2
plt.hist(dice, normed=True)
plt.show()
```











[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:

$$-\sqrt{\ln\left(\sqrt{5}\right)} \cdot \left(10^{\frac{a}{\ln\left(10^4\right)}}\right)$$

Put expression here

```
- math.sqrt(math.log(math.sqrt(5))) * 10.**(aa / math.log(1.e4))
```

Name

MARKS

Part D: Short Answer (5 MARKS)

[5] The code below prints successive versions of a user's word, with hidden letters gradually revealed one letter at a time. Sample output is in the box at the right.

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.

```
 1  def showWord(word, guessed)
        """Create a string showing the word double-spaced,
        with unguessed letters represented by underscores"""
 2      display = ''
 3      for letter in guessed :
 4          if letter in guessed :
 5              display += letter + ' '
 6          else :
 7              display = "_ "
 8      return letter

 9  word = raw_input("Enter a word: ").strip().upper()
10  letters, guessed = word, '', 3
11  while letters :
12      guessed = guessed + letters[0]
13      letters = letters.replace(letters[0] : '')
14      print showWord(word, guessed)
```

Sample output:

```
Enter a word: bananas
B _ _ _ _ _ _
B A _ A _ A _
B A N A N A _
B A N A N A S
```

(1) line 1… ':' is missing at the end the **def**; add it after the closing parenthesis')'
(2) line 3: change 'for letter in guessed' to 'for letter in word'
(3) line 7: change the assignment operator from '=' to '+=', as it is in line 5
(4) line 8: 'return display' instead of 'return letter'
(5) line 10: extra item ,3 on right side of assignment; remove it
(6) line 13: replace ':' by ',' between arguments of letters.replace

Name

MARKS

Extra space

Name

MARKS

Extra space