Answer Key: CMP 230 Final Exam, Version 1, Fall 2014

1. What will the following code print:

```
a = ",Jan,Feb,Mar,Apr,May,Jun,Jul,Aug,Sep,Oct,Nov,Dec,"
b = "Apr 15, 2014"
c = b.split()
print(c)
d = a.split(",")
print(d[1:12])
e = (a.find(c[0]) - 1) / 3
print(e)
f = c[1][:-1]
print(str(int(e)) + "/" + f + "/" + c[2])
```

Answer Key:

```
['Apr', '15,', '2014']
['Jan', 'Feb', 'Mar', 'Apr', 'May', 'Jun', 'Jul', 'Aug', 'Sep', 'Oct', 'Nov']
4.0
4/15/2014
```

2. Write a program to print the fine for speeding. The program must read the speed from user input, then compute and print the fine. The fine is $12 for each mph over 65 and less than or equal to 70, and $15 for each additional mph over 70.

   For example, if the speed is 68 mph, then the fine would be $36 = $12 x 3. If the speed is 72 mph, then the fine would be $90 = $12 x 5 + $15 x 2.

Answer Key:

```
def answer1():
    speed = eval(input("Enter the speed in mph:"))
    if (speed < 65):
        print("No fine")
    else:
        fine = (speed - 65) * 12
        if speed > 70:
            fine = fine + (speed - 70) * 3
        print("The fine is", fine)

def answer2():
    speed = eval(input("Enter the speed in mph:"))
    if speed <= 60:
        print("No fine")
    elif speed <= 70:
        print("The fine is", (speed - 65) * 12)
    else:
        print("The fine is", 60 + (speed - 70) * 15)
```

3. Complete the following program, which reads in a file that has multiple grades, each separated by a comma, and prints out the computed average. That is, write the functions `getGrades()` and `calculateAverage()`:

```
def main():
    grades = getGrades()   #get the file name containing the grades
                           #and return the contents of the file
    avg = calculateAverage(grades) #separate the grades into numbers and compute
                           #the average
    print("The calculated average is:", avg)

main()
```

Answer Key:

```
def getGrades():
    contents = open(input("Enter the file with the grades"), "r").read()
    grades = contents.split(",")
    for i in range(len(grades)):
        grades[i] = int(grades[i])
    return grades
def calculateAverage(grades):
    sum = 0
    for i in grades:
        sum = sum + i
    return sum/len(grades)
```

4. Given the following function definitions:

```
def help(g):
    s = 1
    for h in g:
        s = s + h
        print(s)
    return s

def abc(d):
    e = len(d)
    print("e is ", e)
    if e >= 2:
        f = help(d[0:3])
    elif 2 > e >= 1:
        f = help(d[0:1])
    else:
        f = 5
    return f
```

(a) What does `abc([0,1,2,3])` return?

Answer Key: 4
Write output for partial credit:

Answer Key:

```
e is 4
1
2
4
```

(b) What does `abc([49])` return?

Answer Key: 50
Write output for partial credit:

Answer Key:

```
e is 1
50
```

5. Given the following code:

```
def main():
    file = open("poetry.txt", 'r')
    count = 0
    for line in file:
        line2 = line[:-1] + "?"
        if count % 2 == 0:
            print(line2)
        else:
            print(len(line[:-1]))
        count = count + 1

main()
```

(a) What will the output be for this `poetry.txt`?

poetry.txt:

```
What a
nice
day.
It is.
```

Answer Key:

```
What a?
4
day.?
6
```

(b) What will the output be for this `poetry.txt`?

poetry.txt:

```
No rain
but
cloudy.
```

Answer Key:

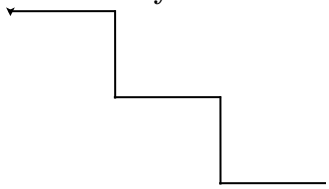
```
No rain?
3
cloudy.?
```

6. (a) Draw what will be displayed in the graphics window when the following program is executed. Remember to indicate the final position and direction of the turtle at the end of program. (The turtle always points to the right of the screen at the start of the program.)

```
from turtle import *

def mystery(t, n):

    for i in range(n):
        t.backward(50)
        if i % 2 == 0:
            t.right(90)
        else:
           t.left(90)

def draw(t, n):
    mystery(t, n)

t = Turtle()
draw(t, 5)
```

Graphics Displayed:

Answer Key:



(b) Write a complete program that asks the user for the radius and color of a circle, creates a graphics window, and draws a circle based on the inputted information. Your `main()` should use functions to complete these tasks, that is, in addition to the `main()` you should write the additional functions `getInput()`, `createWin()`, and `draw()`:

Answer Key:

```
from graphics import *
def main():
    c, r = getInput()    # get the radius and color of a circle
    w = createWin()      # create and return a GraphWin object
    draw(w, c, r)        # draw the circle in the center of w

def getInput():
    col = input("What color should the circle be? "))
    r = eval(input("What is the radius of the circle? "))
    return col, r

def createWin():
    return GraphWin()

def draw(w, c, r):
```

```
        circ = Circle(Point(100, 100), r)
        circ.setFill(r)
        circ.draw(w)
```

7. Write a program that reads in a file called **infile.txt**. For each line in the file it should print out the line followed by "- * -" and then the number of times the lower-case word *the* appears in that line. Finally, it should print out the total number of *the*'s in the file.

   Answer Key:

```
#some comments

def main():
    infile = open('infile.txt')
    totThes = 0
    for line in infile:
        thes = line.count('the')
        totThes += thes
        print(line[:-1] + '- * -' + str(thes))
    print("Total thes:", totThes)

    infile.close()
```

8. Write the Python code for the algorithms below:

   (a)
```
count(ls)
        Set count to 0
        for each item in the list ls
            If item is negative
                increment count
        print count
```

   Answer Key:

```
def count(ls):
    count = 0
    for item in ls:
        If item < 0:
            count = count + 1
    print(count)
```

   (b)
```
search(ls, key, first, last)
        while first is less than last
            Set mid to first + last / 2
            if ls[mid] equals key
                return mid
            else if ls[mid] < key
```

```
            first = mid + 1
        else
            last = mid -1
    return -1
```

Answer Key:

```
def search(ls, key, first, last):
    while first < last:
        mid = (first + last) / 2
        if ls[mid] == key:
            return mid
        elif ls[mid] < key:
            first = mid + 1
        else
            last = mid -1
    return -1
```

9. (a) Write a **complete** class that keeps tracks of information about countries. Your class, `Country` should contain instance variables for the `name`, `population`, `area` and `chocolateProduction`, and should have a constructor method as well as a method, `populationDensity()`, that returns the population density (``population/area'') for the country and a method, `getChocolateProduction()`, that returns the chocolate production for the country.

Answer Key:

```
class Country:
    def __init__(self, name, population, area, chocolateProduction):
        self.name = name
        self.population = population
        self.area = area
        self.chocolateProduction = chocolateProduction

    def populationDensity(self):
        return self.population / self.area

    def getChocolateProduction(self):
        return self.getChocolateProduction
```

(b) Write a function that takes as input a list of `Countries`, called `continent`, and returns the sum of the chocolate production in the list:

```
def overallChocolateProduction(continent):
```

Answer Key:

```
def overallChocolateProduction(continent):
    total = 0
    for c in continent:
```

```
            total = total + c.getChocolateProduction()
        return total
```

10. In the book, a `Student` class and program for finding the student with the highest GPA was designed. Modify the design to also find all full-time high honor students (those with a GPA ≥ 3.75 and currently enrolled in ≥ 12 credits). Your program should print out the name and GPA of all students who meet this criteria. The current credits will be provided as the last entry on each line of the file. Include in your class definition, an instance variable to hold current credits and a new method to access it.

    Clearly mark your changes to the design below:

```
class Student:
    def __init__(self, name, hours, qpoints):
        self.name = name
        self.hours = float(hours)
        self.qpoints = float(qpoints)
    def getName(self):
        return self.name
    def getHours(self):
        return self.hours
    def getQPoints(self):
        return self.qpoints
    def gpa(self):
        return self.qpoints/self.hours

def makeStudent(infoStr):
    # infoStr is a tab-separated line: name hours qpoints
    # returns a corresponding Student object
    name, hours, qpoints = infoStr.split("\t")
    return Student(name, hours, qpoints)

def main():
    # open the input file for reading
    filename = input("Enter name the grade file: ")
    infile = open(filename, 'r')
    # set best to the record for the first student in the file
    best = makeStudent(infile.readline())
    # process subsequent lines of the file
    for line in infile:
        # turn the line into a student record
        s = makeStudent(line)
        # if this student is best so far, remember it.
        if s.gpa() > best.gpa():
            best = s
    infile.close()
    # print information about the best student
    print("The best student is:", best.getName())
    print("hours:", best.getHours())
    print("GPA:", best.gpa())
```

Answer Key:

```
class Student:
    def __init__(self, name, hours, qpoints):          ###Added credits
        self.name = name
        self.hours = float(hours)
        self.qpoints = float(qpoints)
        self.credits = credits                         ###Added credits
    def getName(self):
        return self.name
    def getHours(self):
        return self.hours
    def getQPoints(self):
        return self.qpoints
    def gpa(self):
        return self.qpoints/self.hours
    def getCredits(self)                               ###Added getCredits()
        return self.credits

def makeStudent(infoStr):
    # infoStr is a tab-separated line: name hours qpoints
    # returns a corresponding Student object
    name, hours, qpoints,credits = infoStr.split("\t")       ###Added credits
    return Student(name, hours, qpoints,credits)             ###Added credits

def main():
    # open the input file for reading
    filename = input("Enter name the grade file: ")
    infile = open(filename, 'r')
    # set best to the record for the first student in the file
    best = makeStudent(infile.readline())
    # process subsequent lines of the file
    for line in infile:
        # turn the line into a student record
        s = makeStudent(line)
        # if this student is best so far, remember it.
        if s.gpa() > best.gpa():
            best = s
        if s.gpa() >= 3.75 and s.getCredits() >= 12:               ###Added decision
            print(s.name(), "is a full-time high honor student.")  ###Added print
    infile.close()
    # print information about the best student
    print("The best student is:", best.getName())
    print("hours:", best.getHours())
    print("GPA:", best.gpa())
```

Answer Key: CMP 230 Final Exam, Version 2, Fall 2014

1. What will the following code print:

```
a = ",Jan,Feb,Mar,Apr,May,Jun,Jul,Aug,Sep,Oct,Nov,Dec,"
b = "Mar 15, 2014"
c = b.split()
print(c)
d = a.split(",")
print(d[1:12])
e = a.find(c[0]) / 3
print(e)
f = c[1][:-1]
print(str(int(e)) + "/" + f + "/" + c[2])
```

Answer Key:

```
['Mar', '15,', '2014']
['Jan', 'Feb', 'Mar', 'Apr', 'May', 'Jun', 'Jul', 'Aug', 'Sep', 'Oct', 'Nov']
3.0
3/15/2014
```

2. Write a program to print the fine for speeding. The program must read the speed from user input, then compute and print the fine. The fine is \$10 for each mph over 55 and less than or equal to 65, and \$15 for each additional mph over 65.

   For example, if the speed is 58 mph, then the fine would be \$30 = \$10 x 3. If the speed is 67 mph, then the fine would be \$130 = \$10 x 10 + \$15 x 2.

Answer Key:

```
def answer1():
    speed = eval(input("Enter the speed in mph:"))
    if (speed<55):
        print("No fine")
    else:
        fine = (speed - 55) * 10
        if speed > 65:
            fine = fine + (speed - 65) * 5
        print("The fine is", fine)

def answer2():
    speed = eval(input("Enter the speed in mph:"))
    if speed <= 55:
        print("No fine")
    elif speed <= 65:
        print("The fine is", (speed - 55) * 10)
    else:
        print("The fine is", 100 + (speed - 65) * 15)
```

3. Complete the following program, which reads in a file that has multiple grades, each separated by a semi-colon, and prints out the computed average. That is, write the functions `retrieveGrades()` and `computeAverage()`:

```
def main():
    grades = retrieveGrades()    #get the file name containing the grades
                                 #and return the contents of the file
    avg = computeAverage(grades) #separate the grades into numbers and compute
                                 #the average
    print("The calculated average is:", avg)

main()
```

Answer Key:

```
def retrievetGrades():
    contents = open(input("Enter the file with the grades"), "r").read()
    grades = contents.split(";")
    for i in range(len(grades)):
        grades[i] = int(grades[i])
    return grades
def computeAverage(grades):
    sum = 0
    for i in grades:
        sum = sum + i
    return sum/len(grades)
```

4. Given the following function definitions:

```
def help(g):
    s = 0
    for h in g:
        s = s + 2
        print(s)
    return s

def abc(d):
    e = len(d) - 1
    print("e is", e)
    if e >= 3:
        f = help(d[0:2])
    elif 2 >= e >= 1:
        f = help(d[0:1])
    else:
        f = 10
    return f
```

(a) What does `abc([7,8,9])` return?

Answer Key: 2
Write output for partial credit:

Answer Key:

```
e is 2
2
```

(b) What does `abc([77])` return?

Answer Key: 10
Write output for partial credit:

Answer Key:

```
e is 2
```

5. Given the following code:

```
def main():
    file = open("summary.txt", 'r')
    count = 0
    for line in file:
        line2 = line[:-1] + "?"
        if count % 2 == 0:
            print("count", line2)
        else:
            print(count, line2)
        count = count + 1

main()
```

(a) What will the output be for this `summary.txt`?

summary.txt:

```
It was
short.
```

Answer Key:

```
count It was
1 short.
```

(b) What will the output be for this `summary.txt`?

summary.txt:

```
So
so
short.
```

Answer Key:

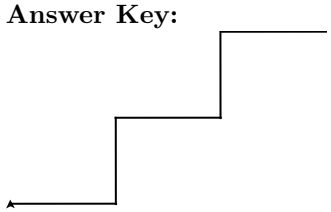
```
count So
1 so
count short.
```

6. (a) Draw what will be displayed in the graphics window when the following program is executed. Remember to indicate the final position and direction of the turtle at the end of program. (The turtle always points to the right of the screen at the start of the program.)

```
from turtle import *

def mystery(t, n):

    for i in range(n):
        t.backward(50)
        if i % 2 == 0:
            t.left(90)
        else:
           t.right(90)

def draw(t, n):
    mystery(t, n)

t = Turtle()
draw(t, 5)
```

Graphics Displayed:

Answer Key:



(b) Write a complete program that asks the user for the outline and fill color of a circle, creates a graphics window, and draws a circle with those colors centered in the window with radius 50. Your `main()` should use functions to complete these tasks, that is, in addition to the `main()` you should write the additional functions `getInput()`, `createWin()`, and `draw()`:

Answer Key:

```
from graphics import *
def main():
    f, o = getInput()    # get the outline and fill color of a circle
    w = createWin()      # create and return a GraphWin object
    draw(w, f, o)        # draw the circle in the center of w with radius 50

def getInput():
    col = input("What color should the circle be filled with? ")
    line = input("What color should the circle be outlined with? ")
    return col, line

def createWin():
    return GraphWin()

def draw(w, fill, out):
    circ = Circle(Point(100, 100), 50)
    circ.setFill(fill)
    circ.setOutline(out)
    circ.draw(w)
```

7. Write a program that reads in a file called **infile.txt**. For each line in the file, the program should print out "- * -" to an output file called **outfile.txt**. Finally, it should print the total number of lines in the input file to the screen.

Answer Key:

```
#some comments

def main():
    infile = open('infile.txt')
    outfile = open('outfile.txt', 'w')
    lines = 0
    for line in infile:
        lines += 1
        print('- * -', file = outfile)
    print("Total lines:", lines)
    outfile.close()
    infile.close()
main()
```

8. Write the Python code for the algorithms below:

   (a)
   ```
   count(ls)
       Set count to 0
       for each item in the list ls
           If item is positive
               increment count
       print count
   ```

   Answer Key:

   ```
   def count(ls):
       count = 0
       for item in ls:
           if item > 0:
               count = count + 1
       print(count)
   ```

   (b)
   ```
   search(ls, key, first, last)
       while first is less than last
           Set mid to first + last / 2
           if ls[mid] is less than key
               Set first to mid + 1
           else
               Set last to mid
       if last equals first and ls[first] equals key
           return first
       else
           return -1
   ```

   Answer Key:

```
def search(ls, key, first, last):
    while first < last:
        mid = (first + last) / 2
        if ls[mid] < key:
            first = mid + 1
        else
            last = mid
    if (last == first) and (ls[first] == key):
        return first
    else
        return -1
```

9. (a) Write a **complete** class that keeps tracks of information about states. Your class, `State` should contain instance variables for the `name`, `numberOfReps`, `population` and `area`, and should have a constructor method as well as a method, `getNumReps()`, that returns the number of representatives for the state and a method, `populationDensity()` that calculates population density (``population/area'').

Answer Key:

```
class State:
    def __init__(self, name, numberOfReps, population, area):
        self.name = name
        self.numberOfReps = numberOfReps
        self.population = population
        self.area = area

    def getNumberOfReps(self):
        return self.numberOfReps

    def populationDensity(self):
        return self.population / self.area
```

(b) Write a function that takes as input a list of `State`s, called `region`, and returns the sum of the number of the representatives for the states in the list:

```
def overallNumReps(region):
```

Answer Key:

```
def overallNumReps(region):
    total = 0
    for state in region:
        total = total + state.getNumberOfReps()
    return total
```

10. In the book, a `Student` class and program for finding the student with the highest GPA was designed. Modify the design to also find all part-time high honor students (those with a GPA ≥ 3.75 and currently enrolled in < 12 credits). Your program should print out the name and GPA of all students who meet this criteria. The current credits will be provided as the last entry on each line of the file. Include in your class definition, an instance variable to hold current credits and a new method to access it.

Clearly mark your changes to the design below:

```
class Student:
    def __init__(self, name, hours, qpoints):
        self.name = name
        self.hours = float(hours)
        self.qpoints = float(qpoints)
    def getName(self):
        return self.name
    def getHours(self):
        return self.hours
    def getQPoints(self):
        return self.qpoints
    def gpa(self):
        return self.qpoints/self.hours

def makeStudent(infoStr):
    # infoStr is a tab-separated line: name hours qpoints
    # returns a corresponding Student object
    name, hours, qpoints = infoStr.split("\t")
    return Student(name, hours, qpoints)

def main():
    # open the input file for reading
    filename = input("Enter name the grade file: ")
    infile = open(filename, 'r')
    # set best to the record for the first student in the file
    best = makeStudent(infile.readline())
    # process subsequent lines of the file
    for line in infile:
        # turn the line into a student record
        s = makeStudent(line)
        # if this student is best so far, remember it.
        if s.gpa() > best.gpa():
            best = s
    infile.close()
    # print information about the best student
    print("The best student is:", best.getName())
    print("hours:", best.getHours())
    print("GPA:", best.gpa())
```

Answer Key:

```
class Student:
    def __init__(self, name, hours, qpoints):          ###Added credits
        self.name = name
        self.hours = float(hours)
        self.qpoints = float(qpoints)
        self.credits = credits                         ###Added credits
    def getName(self):
        return self.name
    def getHours(self):
```

```
        return self.hours
    def getQPoints(self):
        return self.qpoints
    def gpa(self):
        return self.qpoints/self.hours
    def getCredits(self)                                ###Added getCredits()
        return self.credits

def makeStudent(infoStr):
    # infoStr is a tab-separated line: name hours qpoints
    # returns a corresponding Student object
    name, hours, qpoints,credits = infoStr.split("\t")         ###Added credits
    return Student(name, hours, qpoints,credits)             ###Added credits

def main():
    # open the input file for reading
    filename = input("Enter name the grade file: ")
    infile = open(filename, 'r')
    # set best to the record for the first student in the file
    best = makeStudent(infile.readline())
    # process subsequent lines of the file
    for line in infile:
        # turn the line into a student record
        s = makeStudent(line)
        # if this student is best so far, remember it.
        if s.gpa() > best.gpa():
            best = s
        if s.gpa() >= 3.75 and s.getCredits() < 12:             ###Added decision
            print(s.name(), "is a part-time high honor student.") ###Added print
    infile.close()
    # print information about the best student
    print("The best student is:", best.getName())
    print("hours:", best.getHours())
    print("GPA:", best.gpa())
```

Answer Key: CMP 230 Final Exam, Version 3, Fall 2014

1. What will the following code print:

```
a = ",Dec,Nov,Oct,Sep,Aug,Jul,Jun,May,Apr,Mar,Feb,Jan,"
b = "Nov 15, 2014"
c = b.split()
print(c)
d = a.split(",")
print(d[1:12])
e = (a.find(c[0]) - 1) / 4 + 1
print(e)
f = c[1][:-1]
print(str(int(e)) + "/" + f + "/" + c[2])
```

Answer Key:

```
['Nov', '15,', '2014']
['Dec', 'Nov', 'Oct', 'Sep', 'Aug', 'Jul', 'Jun', 'May', 'Apr', 'Mar', 'Feb']
2.0
2/15/2014
```

2. Write a program to print the fine for speeding. The program must read the speed from user input, then compute and print the fine. The fine is \$15 for each mph over 60 and less than or equal to 70, and \$20 for each additional mph over 70.

   For example, if the speed is 63 mph, then the fine would be \$45 = \$15 x 3. If the speed is 72 mph, then the fine would be \$190 = \$15 x 10 + \$20 x 2.

Answer Key:

```
def answer1():
    speed = eval(input("Enter the speed in mph:"))
    if (speed < 60):
        print("No fine")
    else:
        fine = (speed - 60) * 15
        if speed > 70:
            fine = fine + (speed - 70) * 5
        print("The fine is", fine)

def answer2():
    speed = eval(input("Enter the speed in mph:"))
    if speed <= 60:
        print("No fine")
    elif speed <= 70:
        print("The fine is", (speed - 60) * 15)
    else:
        print("The fine is", 150 + (speed - 70) * 20)
```

3. Complete the following program, which reads in a file that has multiple grades, each separated by a colon, and prints out the computed average. That is, write the functions `extractGrades()` and `processAverage()`:

```
def main():
    grades = extractGrades() #get the file name containing the grades
                             #and return the contents of the file
    avg = processAverage(grades) #separate the grades into numbers and compute
                             #the average
    print("The calculated average is:", avg)

main()
```

Answer Key:

```
def extractGrades():
    contents = open(input("Enter the file with the grades"), "r").read()
    grades = contents.split(":")
    for i in range(len(grades)):
        grades[i] = int(grades[i])
    return grades
def processAverage(grades):
    sum = 0
    for i in grades:
        sum = sum + i
    return sum/len(grades)
```

4. Given the following function definitions:

```
def help(g):
    s = 1
    for h in g:
        s = s + 1
        print(s)
    return s

def abc(d):
    e = len(d)
    print("e is ", e)
    if 5 > e > 2:
        f = help(d[0:3])
    elif e > 5:
        f = help(d[2:5])
    else:
        f = 8
    return f
```

(a) What does `abc([10,20,30,40,50,60])` return?

Answer Key: 4 Write output for partial credit:

Answer Key:

```
e is 6
2
3
4
```

(b) What does `abc([5,6,7])` return?

Answer Key: 4 Write output for partial credit:

Answer Key:

```
e is 3
2
3
4
```

5. Given the following code:

```
def main():
    file = open("story.txt", 'r')
    count = 0
    for line in file:
        line2 = "!" + line[:-1]
        if count == 2:
            print(line2)
        else:
            print(line.count("a"))
        count = count + 2

main()
```

(a) What will the output be for this `story.txt`?

story.txt:

```
Once
upon a
time.
```

Answer Key:

```
0
1
!time.
```

(b) What will the output be for this `story.txt`?

story.txt:

```
Here
is
a
story...
```

Answer Key:

```
0
0
!a
0
```

6. (a) Draw what will be displayed in the graphics window when the following program is executed. Remember to indicate the final position and direction of the turtle at the end of program. (The turtle always points to the right of the screen at the start of the program.)

```
from turtle import *

def mystery(t, n):

    for i in range(n):
        t.right(90)
        if i % 2 == 0:
            t.backward(50)
        else:
           t.forward(50)

def draw(t, n):
    mystery(t, n)

t = Turtle()
draw(t, 4)
```

Graphics Displayed:

Answer Key:

(b) Write a complete program that asks the user for the width and height of a rectangle, creates a graphics window, and draws a rectangle with its upper left corner at $(0, 0)$ and the inputted height and width. Your `main()` should use functions to complete these tasks, that is, in addition to the `main()` you should write the additional functions `getInput()`, `createWin()`, and `draw()`:

Answer Key:

```
from graphics import *
def main():
    w, h = getInput()    # get the width and height of a rectangle
    win = createWin()    # create and return a GraphWin object
    draw(win, w, h)      # draw the rectangle with upper left corner at (0, 0)

def getInput():
    w = eval(input("What is the width of the rectangle? "))
    h = eval(input("What's its height? "))
    return w, h

def createWin():
    return GraphWin()

def draw(win, w, h):
```

```
        rect = Rectangle(Point(0, 0), Point(w, h))
        rect.draw(win)
```

7. Write a program that reads in a file called **infile.txt**. For each line in the file it should print out the line followed by "- * -" and then the number of times that the lower-case word *the* appears in that line. Finally, it should print out the total number of times that the lower-case word *the* appears in the file.

Answer Key:

```
#some comments

def main():
    infile = open('infile.txt')
    totThes = 0
    for line in infile:
        thes = line.count('the')
        totThes += thes
        print(line[:-1] + '- * -' + str(thes))
    print("Total thes:", totThes)

    infile.close()
main()
```

8. Write the Python code for the algorithms below:

(a)
```
balance(ls)
        Set balance to 1000
        for each item in the list ls
            Subtract item from balance
        print balance
```

Answer Key:

```
def balance(ls):
    balance = 1000
    for item in ls
        balance = balance - item
    print(balance)
```

(b)
```
search(ls, key, first, last)
        while first is less than last
            Set mid to first + last / 2
            if ls[mid] equals key
                return mid
            else if ls[mid] < key
                first = mid + 1
            else
                last = mid -1
        return -1
```

Answer Key:

```
def search(ls, key, first, last):
    while first < last:
        mid = (first + last) / 2
        if ls[mid] == key
            return mid
        elif ls[mid] < key
            first = mid + 1
        else
            last = mid - 1
    return -1
```

9. (a) Write a **complete** class that keeps tracks of information about boroughs. Your class, `Borough` should contain instance variables for the `name`, `collegeCampuses`, `population` and `area`, and should have a constructor method as well as a method, `getNumCampuses()`, that returns the number of college campuses and a method, `populationDensity()` that calculates population density (‘‘`population/area`’’) for the borough:

Answer Key:

```
class Borough:
    def __init__(self, name, collegeCampuses, population, area):
        self.name = name
        self.collegeCampuses = collegeCampuses
        self.population = population
        self.area = area

    def getNumberCampuses(self):
        return self.collegeCampuses

    def populationDensity(self):
        return self.population / self.area
```

(b) Write a function that takes as input a list of `Boroughss`, called `city`, and returns the sum of the number of the college campuses for the boroughs in the list:

```
def overallCollegeCampuses(city):
```

Answer Key:

```
def overallCollegeCampuses(city):
    total = 0
    for state in city:
        total = total + state.getNumberCampuses()
    return total
```

10. In the book, a `Student` class and program for finding the student with the highest GPA was designed. Modify the design to also find all part-time honor students (those with a GPA ≥ 3.5 and currently enrolled in < 12 credits). Your program should print out the name and GPA of all students who meet this criteria. The current credits will be provided as the last entry on each line of the file. Include in your class definition, an instance variable to hold current credits and a new method to access it.

Clearly mark your changes to the design below:

```
class Student:
    def __init__(self, name, hours, qpoints):
        self.name = name
        self.hours = float(hours)
        self.qpoints = float(qpoints)
    def getName(self):
        return self.name
    def getHours(self):
        return self.hours
    def getQPoints(self):
        return self.qpoints
    def gpa(self):
        return self.qpoints/self.hours

def makeStudent(infoStr):
    # infoStr is a tab-separated line: name hours qpoints
    # returns a corresponding Student object
    name, hours, qpoints = infoStr.split("\t")
    return Student(name, hours, qpoints)

def main():
    # open the input file for reading
    filename = input("Enter name the grade file: ")
    infile = open(filename, 'r')
    # set best to the record for the first student in the file
    best = makeStudent(infile.readline())
    # process subsequent lines of the file
    for line in infile:
        # turn the line into a student record
        s = makeStudent(line)
        # if this student is best so far, remember it.
        if s.gpa() > best.gpa():
            best = s
    infile.close()
    # print information about the best student
    print("The best student is:", best.getName())
    print("hours:", best.getHours())
    print("GPA:", best.gpa())
```

Answer Key:

```
class Student:
    def __init__(self, name, hours, qpoints):          ###Added credits
        self.name = name
        self.hours = float(hours)
        self.qpoints = float(qpoints)
        self.credits = credits                         ###Added credits
    def getName(self):
```

```
        return self.name
    def getHours(self):
        return self.hours
    def getQPoints(self):
        return self.qpoints
    def gpa(self):
        return self.qpoints/self.hours
    def getCredits(self)                                ###Added getCredits()
        return self.credits

def makeStudent(infoStr):
    # infoStr is a tab-separated line: name hours qpoints
    # returns a corresponding Student object
    name, hours, qpoints,credits = infoStr.split("\t")        ###Added credits
    return Student(name, hours, qpoints,credits)             ###Added credits

def main():
    # open the input file for reading
    filename = input("Enter name the grade file: ")
    infile = open(filename, 'r')
    # set best to the record for the first student in the file
    best = makeStudent(infile.readline())
    # process subsequent lines of the file
    for line in infile:
        # turn the line into a student record
        s = makeStudent(line)
        # if this student is best so far, remember it.
        if s.gpa() > best.gpa():
            best = s
        if s.gpa() >= 3.5 and s.getCredits() < 12:          ###Added decision
            print(s.name(), "is a part-time honor student.") ###Added print
    infile.close()
    # print information about the best student
    print("The best student is:", best.getName())
    print("hours:", best.getHours())
    print("GPA:", best.gpa())
```

Answer Key: CMP 230 Final Exam, Version 4, Fall 2014

1. What will the following code print:

```
a = ",Dec,Nov,Oct,Sep,Aug,Jul,Jun,May,Apr,Mar,Feb,Jan,"
b = "Oct 15, 2014"
c = b.split()
print(c)
d = a.split(",")
print(d[1:12])
e = a.find(c[0]) / 3
print(e)
f = c[1].rstrip(",")
print(str(int(e)) + "/" + f + "/" + c[2])
```

Answer Key:

```
['Oct', '15,', '2014']
['Dec', 'Nov', 'Oct', 'Sep', 'Aug', 'Jul', 'Jun', 'May', 'Apr', 'Mar', 'Feb']
3.0
3/15/2014
```

2. Write a program to print the fine for speeding. The program must read the speed from user input, then compute and print the fine. The fine is \$15 for each mph over 45 and less than or equal to 65, and \$25 for each additional mph over 65.

   For example, if the speed is 48 mph, then the fine would be \$45 = \$15 x 3. If the speed is 67 mph, then the fine would be \$350 = \$15 x 20 + \$25 x 2.

Answer Key:

```
def answer1():
    speed = eval(input("Enter the speed in mph:"))
    if (speed < 45):
        print("No fine")
    else:
        fine = (speed - 45) * 15
        if speed > 65:
            fine = fine + (speed - 65) * 10
        print("The fine is", fine)

def answer2():
    speed = eval(input("Enter the speed in mph:"))
    if speed <= 45:
        print("No fine")
    elif speed <= 65:
        print("The fine is", (speed - 45) * 15)
    else:
        print("The fine is", 300 + (speed - 65) * 25)
```

3. Complete the following program, which reads in a file that has multiple grades, each separated by a hyphen, and prints out the computed average. That is, write the functions `acquireGrades()` and `determineAverage()`:

```
def main():
    grades = acquireGrades() #get the file name containing the grades
                             #and return the contents of the file
    avg = determineAverage(grades) #separate the grades into numbers and compute
                             #the average
    print("The calculated average is:", avg)

main()
```

Answer Key:

```
def acquireGrades():
    contents = open(input("Enter the file with the grades"), "r").read()
    grades = contents.split("-")
    for i in range(len(grades)):
        grades[i] = int(grades[i])
    return grades
def determineAverage(grades):
    sum = 0
    for i in grades:
        sum = sum + i
    return sum/len(grades)
```

4. Given the following function definitions:

```
def help(g):
    s = 0
    for h in g:
        s = s + h
        print(s)
    return s

def abc(d):
    e = len(d) + 1
    print("e is ", e)
    if e >= 3:
        f = help(d[0:2])
    elif 3 > e >= 2:
        f = help(d[0:1])
    else:
        f = 65
    return f
```

(a) What does `abc([2,2])` return?

Answer Key: 4
Write output for partial credit:

Answer Key:

```
e is 3
2
4
```

(b) What does `abc([99])` return?

Answer Key: 99
Write output for partial credit:

Answer Key:

```
e is 2
99
```

5. Given the following code:

```
def main():
    file = open("poetry.txt", 'r')
    count = 0
    for line in file:
        line2 = line[:-1] + "?"
        if count % 2 == 0:
            print(line2)
        else:
            print(len(line[:-1]))
        count = count + 1

main()
```

(a) What will the output be for this `poetry.txt`?

poetry.txt:

```
What a
nice
day.
It is.
```

Answer Key:

```
What a?
4
day.?
6
```

(b) What will the output be for this `poetry.txt`?

poetry.txt:

```
No rain
but
cloudy.
```

Answer Key:

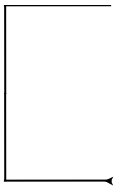
```
No rain?
3
cloudy.?
```

6. (a) Draw what will be displayed in the graphics window when the following program is executed. Remember to indicate the final position and direction of the turtle at the end of program. (The turtle always points to the right of the screen at the start of the program.)

```
from turtle import *

def mystery(t, n):

    for i in range(n):
        t.right(90)
        if i % 2 == 0:
            t.backward(50)
        else:
           t.right(90)
           t.forward(50)

def draw(t, n):
    mystery(t, n)

t = Turtle()
draw(t, 4)
```

Graphics Displayed:

Answer Key:



(b) Write a complete program that asks the user for the width and color of a square, creates a graphics window, and draws a square with its lower right corner at $(199, 199)$ and width and color based on the inputted information. Your `main()` should use functions to complete these tasks, that is, in addition to the `main()` you should write the additional functions `getInput()`, `createWin()`, and `draw()`:

Answer Key:

```
from graphics import *
def main():
    w, c = getInput()    # get the width and color of a square
    win = createWin()    # create and return a GraphWin object
    draw(win, w, c)      # draw the square with the lower right corner at (199, 199)

def getInput():
    col = input("What color should the square be? "))
    width = eval(input("What is the width of the square? "))
    return width, col

def createWin():
    return GraphWin()
```

```
    def draw(win, w, c):
        square = Rectangle(Point(199 - w, 199 - w), Point(199, 199))
        square.setFill(c)
        square.draw(w)
```

7. Write a program that reads in a file called **infile.txt**. For each line in the file, the program should print out the line followed by ": " and then the number of times that the lower-case letter *a* appears in that line. Finally, it should print out the total number of times the lower-case letter *a* appear in the file.

   Answer Key:

```
#some comments

def main():
    infile = open('infile.txt')
    totAs = 0
    for line in infile:
        as = line.count('a')
        totAs += as
        print(line[:-1] + ':', as)
    print("Total as:", totAs)

    infile.close()
main()
```

8. Write the Python code for the algorithms below:

   (a)
```
total(ls)
    Set total to 0
    for each item in the list ls
        Add item to total
    print total
```

   Answer Key:

```
def total(ls):
    total = 0
    for item in ls:
        total = total + item
    print(total)
```

   (b)
```
search(ls, key, first, last)
    while first is less than last
        Set mid to first + last / 2
        if ls[mid] is less than key
            Set first to mid + 1
        else
```

```
            Set last to mid
        if last equals first and ls[first] equals key
            return first
        else
            return -1
```

Answer Key:

```
def search(ls, key, first, last):
    while first < last:
        mid = (first + last)/ 2
        if ls[mid] < key:
            first = mid + 1
        else
            last = mid
    if (last == first) and (ls[first] == key):
        return first
    else
        return -1
```

9. (a) Write a **complete** class that keeps tracks of information about train lines. Your class, `TrainLine` should contain instance variables for the `name`, `length`, `dailyRidership` and `coverageArea`, and should have a constructor method as well as a method, `getLength()`, that returns the train length a method, `riderDensity()` that calculates rider density (``dailyRidership/coverageArea''):

Answer Key:

```
class TrainLine:
    def __init__(self, name, length, dailyRidership, coverageArea):
        self.name = name
        self.length = length
        self.dailyRidership = dailyRidership
        self.coverageArea = coverageArea

    def getLength(self):
        return self.length

    def riderDensity(self):
        return self.dailyRidership / self.coverageArea
```

(b) Write a function that takes as input a list of `TrainLine`s, called `subway`, and returns the sum of the length of the train lines in the list:

```
def overallLength(subway):
```

Answer Key:

```
def overallLength(subway):
    total = 0
```

```
        for line in subway:
            total = total + line.getLength()
        return total
```

10. In the book, a `Student` class and program for finding the student with the highest GPA was designed. Modify the design to also find all full-time honor students (those with a GPA $\geq$ 3.5 and currently enrolled in $\geq$ 12 credits). Your program should print out the name and GPA of all students who meet this criteria. The current credits will be provided as the last entry on each line of the file. Include in your class definition, an instance variable to hold current credits and a new method to access it.

    Clearly mark your changes to the design below:

```
class Student:
    def __init__(self, name, hours, qpoints):
        self.name = name
        self.hours = float(hours)
        self.qpoints = float(qpoints)
    def getName(self):
        return self.name
    def getHours(self):
        return self.hours
    def getQPoints(self):
        return self.qpoints
    def gpa(self):
        return self.qpoints/self.hours

def makeStudent(infoStr):
    # infoStr is a tab-separated line: name hours qpoints
    # returns a corresponding Student object
    name, hours, qpoints = infoStr.split("\t")
    return Student(name, hours, qpoints)

def main():
    # open the input file for reading
    filename = input("Enter name the grade file: ")
    infile = open(filename, 'r')
    # set best to the record for the first student in the file
    best = makeStudent(infile.readline())
    # process subsequent lines of the file
    for line in infile:
        # turn the line into a student record
        s = makeStudent(line)
        # if this student is best so far, remember it.
        if s.gpa() > best.gpa():
            best = s
    infile.close()
    # print information about the best student
    print("The best student is:", best.getName())
    print("hours:", best.getHours())
    print("GPA:", best.gpa())
```

Answer Key:

```
class Student:
    def __init__(self, name, hours, qpoints):          ###Added credits
        self.name = name
        self.hours = float(hours)
        self.qpoints = float(qpoints)
        self.credits = credits                         ###Added credits
    def getName(self):
        return self.name
    def getHours(self):
        return self.hours
    def getQPoints(self):
        return self.qpoints
    def gpa(self):
        return self.qpoints/self.hours
    def getCredits(self)                               ###Added getCredits()
        return self.credits

def makeStudent(infoStr):
    # infoStr is a tab-separated line: name hours qpoints
    # returns a corresponding Student object
    name, hours, qpoints,credits = infoStr.split("\t")       ###Added credits
    return Student(name, hours, qpoints,credits)             ###Added credits

def main():
    # open the input file for reading
    filename = input("Enter name the grade file: ")
    infile = open(filename, 'r')
    # set best to the record for the first student in the file
    best = makeStudent(infile.readline())
    # process subsequent lines of the file
    for line in infile:
        # turn the line into a student record
        s = makeStudent(line)
        # if this student is best so far, remember it.
        if s.gpa() > best.gpa():
            best = s
        if s.gpa() >= 3.5 and s.getCredits() >= 12:          ###Added decision
            print(s.name(), "is a full-time honor student.") ###Added print
    infile.close()
    # print information about the best student
    print("The best student is:", best.getName())
    print("hours:", best.getHours())
    print("GPA:", best.gpa())
```