Answer Key: CMP 167 Final Exam, Version 3, Spring 2015

1. What will the following code print:

```
s = "haskell::curry::utrecht::glasgow"
a = s[0:3]
print(a.upper())
names = s.split("::")
print(names)
b,c,d = names[1],names[2],names[3]
print(c,d)
print(a[0] + b.upper())
print("versions:", c.title(),d.capitalize())
print('main = putStrLn "', names[0],'"')
```

Answer Key:

```
HAS
['haskell', 'curry', 'utrecht', 'glasgow']
utrecht glasgow
hCURRY
versions: Utrecht Glasgow
main = putStrLn " haskell "
```

2. Write a **complete program** to calculate how much something will weigh on Jupiter. Your program should prompt the user for the weight on the Earth and then print out the weight on Jupiter. For example, if the user enters 100, your program should print out 254.

   The weight of an item on Jupiter is 254% *of its weight on earth.*

Answer Key:

```
#Computes weights on Jupiter
def main():
     earthWeight = eval(input('Enter earth weight: '))
     jupiterWeight = earthWeight * 2.54
     print('The weight on Jupiter is:', jupiterWeight)

main()
```

3. What is output of the code below:

```
def prob4(paul, john):
     if paul > 2:
          print("Easy case")
          yoko = -1
     else:
          print("Complex case")
          yoko = helper(paul,john)
     return(yoko)
```

```
def helper(ringo,george):
     s = ""
     for j in range(ringo):
          print(j, ": ", george[j])
          if j % 2 == 0:
               s = s + george[j]
               print("Building s:", s)
     return(s)
```

(a)
```
r = prob4(6, "city")
print("Return:  ", r)
```

Output:

Answer Key:

```
Small case
Return: -1
```

(b)
```
r = prob4(2,"university")
print("Return:  ", r)
```

Output:

Answer Key:

```
Complex case
0 :  u
Building s: u
1 :  n
Return:  u
```

(c)
```
r = prob4(4,"new york")
print("Return:  ", r)
```

Output:

Answer Key:

```
Complex case
0 :  n
Building s: n
1 :  e
2 :  w
Building s: nw
3 :
Return:  nw
```

4. Given the following program and input file, what is printed:

```
def prob5V1():
   c = 0
   infile=open("places.txt","r")
   for line in infile.readlines():
      if len(line) > 9:
         print("Long Line: ", end ="")
         c = c + 1
      print(line)
   print("Num long lines is", c)

prob5V1()
```

places.txt

```
Santo Domingo
Santiago
San Cristobal
Puerta Plata
La Vega
La Altagracia
```

Output:

Answer Key:

```
Long Line: Santo Domingo

Santiago

Long Line: San Cristobal

Long Line: Puerta Plata

La Vega

Long Line: La Altagracia

Num long lines is 4
```

5. (a) Write a function that takes number between 1 and 7 as a parameter and returns the corresponding number as a string. For example, if the parameter is 1, your function should return `"one"`. If the parameter is 2, your function should `"two"`, etc. If the parameter is not between 1 and 7, your function should return the empty string.

Answer Key:

```
def returnNumString(num):
    if num == 1:
        return "one"
    elif num == 2:
        return "two"
    elif num == 3:
        return "three"
    elif num == 4:
        return "four"
    elif num == 5:
        return "five"
    elif num == 6:
        return "six"
    elif num == 7:
        return "seven"

    else:
        return ""
```

(b) Write a `main()` that allows the user to enter a number and calls your function to show that it works.

Answer Key:

```
#intro comment
def main():
    num = eval(input("Enter a number"))
```

```
        test1 = returnNumString(num)
        print ("Testing my function:", num, "is", test1)
    main()
```

6. Complete the following program, which sets up a graphics window and turtle, draws two squares to the window, and then prints a closing message and closes the graphics window when mouse is clicked. That is, write the functions `setUp()`, `draw2Squares()`, and `conclusion()`:

```
import turtle

def main():
    w,t = setUp()   #sets up a graphics window and turtle
    draw2Squares(t)  #draws 2 squares using the turtle
    conclusion(w)   #prints goodbye and closes window on click

main()
```

Answer Key:

```
def setUp():
    trey = turtle.Turtle()
    win = turtle.Screen()
    return(win,trey)

def draw2Squares(t):
    for i in range(4):
        t.forward(100)
        t.right(360/4)
    left(90)
    for i in range(4):
        t.forward(100)
        t.right(360/4)

def conclusion(w):
    print("Goodbye!")
    w.exitonclick()
```

7. (a) Write a **complete** program that prompts the user for a file name and prints the number of lines in the file.

Answer Key:

```
#some comments

def main():
    fileName = input('Enter file name: ')
    infile = open(fileName)
    data = infile.read()
    print("Number of lines:", data.count("\n"))
```

```
infile.close()
```

(b) Write a **complete** program that prints the population stored in a data file. Your program should open the file, `cityData.csv` and sum the last values in the lines (the populations). Note that the first line should not be used since it contains the column headers and not data. The data is separated by commas (“,”). Your program should **print** the running sum that you calculated.

cityData.csv:

```
Borough, County, Population
Bronx, Bronx, 1385108
Brooklyn, Kings, 2504700
Manhattan, New York, 1585873
Queens, Queens, 2230722
Staten Island, Richmond, 468730
```

Answer Key:

```
#some comments

def main():
    sum = 0
    infile = open("population.csv")
    infile.readline()   #Ignore first line, since no numbers
    lines = infile.readlines()
    for l in lines:
        cells = l.split(',')
        sum = sum + eval(cells[2])

    print("Total population:", sum)

    infile.close()
```

8. Write the Python code for the algorithms below:

(a)
```
getInput()
Ask user for a positive number
Until they enter a positive number
    Print error message
    Ask user for a positive number
Return the positive number entered
```

Answer Key:

```
def getInput()
    x = int(eval('Enter a positive number: '))
    while x <= 0:
        print('Not an positive number!')
        x = int(eval('Enter a positive number: '))
    return(x)
```

(b)
```
sort(ls)
    Set L to be the length of the list ls.
    For i = 0,1,...,L-2:
        For j = 0,1,...,L-2:
            If ls[j] is bigger than ls[j+1], swap the values
    Return the list, ls.
```

Answer Key:

```
def sort(ls):
    L = len(ls)
    for i in range(L-1):
        for j in range(L-1):
            if ls[j] > ls[j+1]:
                ls[j],ls[j+1] = ls[j+1],ls[j]
    return ls
```

9. In lab, we wrote a Tic-Tac-Toe program. Modify the program to stop the game when someone has won. Your program should check for a winner each move. Your program should continue playing until there is a winner or until all squares are filled.

   Clearly mark your changes to the design below:

```
#Second Version of Tic-Tac-Toe
from turtle import *
def setUp():
    win, tic = Screen(), Turtle()
    tic.speed(10)
    win.setworldcoordinates(-0.5,-0.5,3.5, 3.5)
    for i in range(1,3):
        tic.up()
        tic.goto(0,i)
        tic.down()
        tic.forward(3)
    tic.left(90)
    for i in range(1,3):
        tic.up()
        tic.goto(i,0)
        tic.down()
        tic.forward(3)
    tic.up()
    board = [["","",""],["","",""],["","",""]]
    return(win,tic,board)
def playGame(tic,board):
    for i in range(4):
        x,y = eval(input("Enter x, y coordinates for X's move: "))
        tic.goto(x+.25,y+.25)
        tic.write("X",font=('Arial', 90, 'normal'))
        board[x][y] = "X"
        x,y = eval(input("Enter x, y coordinates for O's move: "))
        tic.goto(x+.25,y+.25)
        tic.write("O",font=('Arial', 90, 'normal'))
```

```
        board[x][y] = "O"
    x,y = eval(input("Enter x, y coordinates for X's move: "))
    tic.goto(x+.25,y+.25)
    tic.write("X",font=('Arial', 90, 'normal'))
    board[x][y] = "X"
def checkWinner(board):
    for x in range(3):
        if board[x][0] != "" and (board[x][0] == board[x][1] == board[x][2]):
            return(board[x][0])  #we have a non-empty row that's identical
    for y in range(3):
        if board[0][y] != "" and (board[0][y] == board[1][y] == board[2][y]):
            return(board[0][y])  #we have a non-empty column that's identical
    if board[0][0] != "" and (board[0][0] == board[1][1] == board[2][2]):
        return(board[0][0])
    if board[2][0] != "" and (board[2][0] == board[1][1] == board[2][0]):
        return(board[2][0])
    return("No winner")
def main():
    win,tic,board = setUp()   #Set up the window and game board
    playGame(tic,board)       #Ask the user for the moves and display
    print("\nThe winner is", checkWinner(board))  #Check for winner
```

Answer Key:

```
#Second Version of Tic-Tac-Toe
from turtle import *
def setUp():
    win, tic = Screen(), Turtle()
    tic.speed(10)
    win.setworldcoordinates(-0.5,-0.5,3.5, 3.5)
    for i in range(1,3):
        tic.up()
        tic.goto(0,i)
        tic.down()
        tic.forward(3)
    tic.left(90)
    for i in range(1,3):
        tic.up()
        tic.goto(i,0)
        tic.down()
        tic.forward(3)
    tic.up()
    board = [["","",""],["","",""],["","",""]]
    return(win,tic,board)
def playGame(tic,board):
    numMoves = 0                                    ###ADDED
    while checkWinner == "No Winner" and numMoves < 9:###ADDED
        numMoves += 1                               ###ADDED
        if numMoves % 2 == 0:                       ###ADDED
            x,y = eval(input("Enter x, y coordinates for X's move: "))
            tic.goto(x+.25,y+.25)
            tic.write("X",font=('Arial', 90, 'normal'))
```

```
            board[x][y] = "X"
        else:                                       ###ADDED
            x,y = eval(input("Enter x, y coordinates for O's move: "))
            tic.goto(x+.25,y+.25)
            tic.write("O",font=('Arial', 90, 'normal'))
            board[x][y] = "O"
    if checkWinner != "No Winner":                  ###ADDED
        print("There was a winner!")                ###ADDED
    else:                                           ###ADDED
        print("Game Over:  No winner!")             ###ADDED
def checkWinner(board):
    for x in range(3):
        if board[x][0] != "" and (board[x][0] == board[x][1] == board[x][2]):
            return(board[x][0])  #we have a non-empty row that's identical
    for y in range(3):
        if board[0][y] != "" and (board[0][y] == board[1][y] == board[2][y]):
            return(board[0][y])  #we have a non-empty column that's identical
    if board[0][0] != "" and (board[0][0] == board[1][1] == board[2][2]):
        return(board[0][0])
    if board[2][0] != "" and (board[2][0] == board[1][1] == board[2][0]):
        return(board[2][0])
    return("No winner")
def main():
    win,tic,board = setUp()   #Set up the window and game board
    playGame(tic,board)       #Ask the user for the moves and display
    print("\nThe winner is", checkWinner(board))  #Check for winner
```

10. (a) Write a **complete** class that keeps tracks of information about apartment. Your class, `Apartment` should contain instance variables for the `apartmentNumber`, `rent`, `area` and `floor`, and should have a constructor method as well as a method, `pricePerSquareFoot()`, that returns the price (`rent/area`) for the price per square foot of the apartment and a method, `getFloor()`, that returns the floor on which the apartment is located.

Answer Key:

```
class Apartment:
    def __init__(self, apartmentNumber, rent, area, floor):
        self.apartmentNumber = apartmentNumber
        self.rent = rent
        self.area = area
        self.floor = floor

    def pricePerSquareFoot(self):
        return self.rent / self.area

    def getFloor(self):
        return self.floor
```

(b) Write a function that takes as input a list of `Apartment`s, called `building`, and the best value apartment in the building (i.e. the minimum of all the pricePerSquareFoot of the apartments in the inputted list):

```
    def bestValue(building):
```

Answer Key:

```
def bestValue(building):
    best = 0
    for a in building:
        if a.pricePerSquareFoot() < best:
            best = a.pricePerSquareFoot()
    return best
```