Answers/Outputs for Questions

1) a) Called with 70, 30 and random seed 57.

```python
def game(ra, rb):
    global scoreA
    global scoreB
    global plotPointsNormal

    prob = float(ra) / (ra + rb) # Standard probability calculation

    if (scoreA >= 11 or scoreB >= 11) and math.fabs(scoreA - scoreB) >= 2:
        # Checks if someone has got to 11 or greater, and they win by 2 clear points
        # Using math.fabs to check if the leader is winning by 2 clear points
        result = (scoreA, scoreB)
        scoreA = scoreB = 0
        return result

    if random.random() < prob:
    # Pseudo-random scoring system weighted on the probability that adds 1 each time
        scoreA += 1
    else:
        scoreB += 1

    return game(ra, rb)
```

```
pc-67-106:PythonAssessment1 niallcurtis$ python game.py
(11, 5)
```

1) b) Called with ra=70, rb= 30, and number of games 5000.

```python
def winProbability(ra, rb, n):
    winsA = 0

    for i in range(n):
        # Run a simulation for n and add 1 to the count whenever A wins
        result = game(ra, rb)
        winsA += result[0] > result[1]

    return float(winsA) / n
```

```
pc-67-106:PythonAssessment1 niallcurtis$ python game.py
0.98
```

1) c) Called with test.csv

```python
def readList(file):
    with open(file) as csvfile:
        reader = csv.reader(csvfile)
        next(reader)

        return [(int(row[0]), int(row[1])) for row in reader]
```

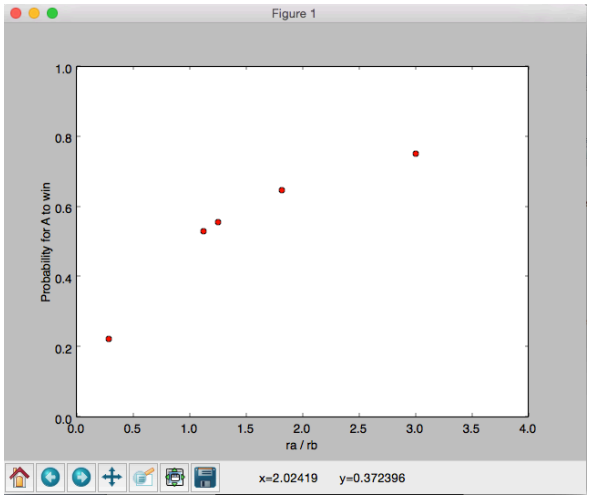
```
pc-67-106:PythonAssessment1 niallcurtis$ python game.py
[(60, 20), (100, 55), (50, 40), (20, 70), (95, 85)]
```

1) d)

```python
def makePlot(probList):
    probabilities = [float(row[0]) / (row[0] + row[1]) for row in probList]
    # Calculate probabilites from the csv
    skill = [float(row[0]) / row[1] for row in probList]
    # Same but with skill

    plt.plot(skill, probabilities, 'ro')
    # Standard plotting stuff, setting up graph

    plt.axis([0, 4, 0, 1])
    plt.xlabel('ra / rb')
    plt.ylabel('Probability for A to win')
    plt.show()
```



1) e) Called with ra=60, rb=

```python
def simulateUntilProbability(ra, rb, desiredProb):
    i = 0
    prob = 0 # Calculate base probability of A winning
    # As long as the probability of a winning "i" games is lower than the desired probability
    # increase count of games played
    while prob <= desiredProb:
        i += 1
        fact = math.factorial # Simplyifying things
        calculateFact = (fact(2 * i - 1)) / ((fact(i)) * fact(i - 1))
        binomial = (calculateFact * (0.6 ** i)) * 0.4 ** (i - 1) # Binomial expansion
        prob = prob + binomial

    return i
```

```
pc-67-106:PythonAssessment1 niallcurtis$ python game.py
2
```

2) More complex, more screenshots. Includes code for both the English game simulation, along with the plotting system for both types of scoring.
Function has been called using the playerSkillsEx list, with 50 games tested for each skill matchup.

```python
playerSkillsEx = [(1, 99), (10, 90), (20, 80), (30, 70), (40, 60), (50, 50)] # Can't divide by zero!
def makeTimePlot(probList, nGames): # List of skill tuples and number of games to simulate
    global numberOfRallies
    # Assumptions:
    # Due to the fact that in PARS scoring, every score is exactly one rally, and in
    # English scoring one score can involve many rallies, the assumption is made that
    # an English game will take an increasingly long time the closer the skill is between players as
    # there is more chance that one point has multiple rallies.
    # Theoretically in a simulation if one player has an significantly higher skill than another,
    # an English game would be faster than a PARS game as it is to 9 points not 11.
    playerProbabilities = []
    lenNormalAvg = [] #Average lists to add to
    lenEnglishAvg = []
    for i in probList:
        playerProbabilities.append(float(i[0]) / i[1]) # Calculate ra / rb
        lenNormal = [] # Lists for Averages
        for n in range(nGames):
            lenNormal.append(game(i[0], i[1])) # Simulate games and append list
            englishGame(i[0], i[1]) # We don't want the total score added for the English game,
            # because there can be more rallies than points. Thus, we use a different method.
        numberOfRallies = numberOfRallies / nGames # Mean number of rallies
        lenEnglishAvg.append(numberOfRallies) # Add number to list
        numberOfRallies = 0 # Reset for next pair
        lenNormalAvg.append(sum([i*j for i, j in lenNormal]) / len(lenNormal)) # Calculate mean average of games lengths

    plt.plot(playerProbabilities, lenNormalAvg, label = "PARS Scoring")
    plt.plot(playerProbabilities, lenEnglishAvg, label = "English Scoring")

    plt.legend(loc = 'upper left')
    plt.xlabel('Player Skill ra/rb') # Assuming one rally is 10 seconds
    plt.ylabel('Game Time in Seconds')
    plt.title('Probability of Player Winning / Game time over ' + str(nGames) + ' games')
    plt.show()
```

```python
playUntilDecision = 0
numberOfRallies = 0
def englishGame(ra, rb, isA = True, nRallies = 0):
    global scoreA
    global scoreB
    global playUntilDecision
    global numberOfRallies

    numberOfRallies += 1 # Counting number of rallies for future question

    prob = float(ra) / (ra + rb)
    prob = prob if isA else 1 - prob # Probability calculation that can be changed if initial server is B

    # We neglect who hits 8 first for the sake simplicity because either way playUntilDecision is random
    if playUntilDecision == 0 and scoreA == 8 and scoreB == 8:
        playUntilDecision = random.randint(9, 10)

    # We play fixed until 9 if we haven't hit 8-8
    playUntil = 9 if playUntilDecision == 0 else playUntilDecision

    # Finishing the game if a player meets the win condition
    if (scoreA >= playUntil or scoreB >= playUntil):
        result = (scoreA, scoreB)
        scoreA = scoreB = playUntilDecision = 0
        return result

    # Simple scoring system like PARS system, except with the exception to switch instead of adding a point
    if random.random() < prob:
        if isA:
            scoreA += 1
        else:
            scoreB += 1
    else:
        isA = not isA

    return englishGame(ra, rb, isA, nRallies + 1)
```