CSI31 Introduction to Computer Programming I

Dr. Sharon Persinger
November 21, 2018

Topics

- Simulation
- Randomness and pseudorandom number generator functions in Python
- Racquetball
- Top-down design

Simulation and randomness

- Simulation: a representation through a computer program of some real event – business, town, battle, science experiment.
- Called Monte Carlo simulations when the simulation uses a pseudo-random number generator to generate numbers with uncertainty

- Randomness: Python's pseudorandom number module random
-
- randrange(start, stop) chooses a pseudorandom int x that satisfies start <=x < stop

- random() chooses a pseudorandom float x that satisfies 0 <=x < 1

- Both uniform distribution: all possible values are equally likely to be returned by the function
- Import the functions from the module random when you need them

Examples

- Write a function that returns a tuple that is the result of rolling two dice.
- Write a function that returns a random card from a standard 52 card deck.
- Write a function that returns True randomly p of the time. Here p is a number between 0 and 1 inclusive.

Racquetball

- We are going to write a simulation of the game of racquetball.

- http://www.youtube.com/watch?v=EXvyNKaFkaU

Racquetball

- Racquet sport played with a short-handled racquet, hollow ball, on a court with four walls. Like handball but with a racquet.
- Play: Server puts the ball into play. Players alternate hitting the ball to keep it in play legally – rally. Player who fails to hit the ball loses the rally. If the server wins the rally, a point is won. If the server loses the rally, serve goes to the other player. In order to win a point, a player must be serving.
- Scoring: The first person to win 15 points wins the game.

Racquetball simulation

- What is the effect of small differences in ability in racquetball? Measure the differences in ability by probability of winning a serve.
- Program specification:
  - Input: Program gets as input the service probabilities for Player A and Player B and the number of games to simulate.
  - Output: After the program has done the simulation, it prints a report showing the number of games simulated and the number and percent of games won by each player.
- The specification tells what the program should do, not how it will do that.

Top-down design

- Start with the general problem.
- Express that problem in terms of smaller problems.
- Then express those problems in terms of smaller problems, and so on, until you have described a small, simple problem that you can write a program to solve.
- Also called successive refinement.

- Encourages the programmer to break a problem into simpler parts
- Encourages the programmer to think about solving one simpler problem at a time

Top level design for Racquetball simulation

- Print an introduction
- Get the inputs: probA, probB, n
- Simulate n games of racquetball using probA, probB.
- Print a report on the wins for playerA, playerB.

- THINK ABOUT ONE TASK AT A TIME!

Convert this immediately into a main program with functions

```
def main():
    printIntro():
    probA, probB, n = getInputs()
    winsA, winsB = SimNgames(n, probA, probB)
    printSummary(winsA, winsB)
```

We’ve decided what functions we need to write, what parameters they take, how many values they return. The variable names indicate the roles of the parameters and the return values.

Implementation

- Some functions we can implement immediately:
  - printIntro()
  - getInputs()
  - printSummary(winsA, winsB)
-

printIntro()

```
def printIntro():
        print("This program simulates a game of racquetball between two")
        print('players called "A" and "B". The  ability of each player is')
        print("indicated by a probability (a number between 0 and 1) that")
        print("the player wins the point when serving. Player A always")
        print("has the first serve.")
```

getInputs()

```
def getInputs():
#Returns the three simulation parameters
   a = eval(input("What is the prob. player A wins a serve? "))
   b = eval(input("What is the prob. player B wins a serve? "))
   n = eval(input("How many games to simulate? "))

   return a, b, n
```

printSummary(winsA, winsB)

```
def printSummary(winsA, winsB):
#Prints a summary of wins for each player.
          n = winsA + winsB
          print("\nGames simulated:", n)
          print("Wins for A: {0}({1:0.1%})".format(winsA, winsA/n))
          print("Wins for B: {0} {1:0.1%})".format(winsB, winsB/n))
```

Formatted output

- str.format(**args*, ***kwargs*) Perform a string formatting operation. Read section 5.8

- Many different types of string formatting operations
- {0}: replace this with the 0-position argument
- ({1:0.1%}): replace with the 1-postion argument, formatted by 0.1%, which means formatted as a percent with 1 decimal place

What’s left? Designing simNGames

Counted loop to simulate one game, n times

```
Initialize winsA and WinsB to 0
Loop n times
    Simulate a game  #still have to do this
    If playerA wins
        Add 1 to winsA
    Else
        Add 1 to winsB
```

simNGames

```
def simNGames(n, probA, probB):
#Simulates n games of racquetball between players
#whose abilities are represented by the probability of
 #winning a serve. Returns number of wins for A and B
winsA = winsB = 0
 for i in range(n):
        scoreA, scoreB = simOneGame(probA, probB)
                              #still have to do this
        if scoreA > scoreB:
                winsA = winsA + 1
        else:
                winsB = winsB + 1
return winsA, winsB
```

Third-level design: simOneGame

Design:

```
 Initialize scores to 0
 Set serving to 'A'
 Loop while game is not over:
          Simulate one serve of whichever player is serving
         Update the status- score and player serving- of the
         game depending on  outcome of serve
Return scores
```

Easy part of simOneGame

```
def simOneGame(probA, probB):
    scoreA = 0
    scoreB = 0
    serving = 'A'
    while not gameOver(scoreA, scoreB) #still have to do this
        if serving == 'A':
             if random() < probA:
                scoreA = score A+1
            else:
                serving == 'B”
        else:  #serving == ‘B’
            if random() < probB:
                scoreB = score B+1
            else:
                serving == 'A”
    return score A,  scoreB
```

What does

```
if random() < probA:
        scoreA = score A+1
 else:
        serving == 'B”
```

Do?

Last part: gameOver()

```
def gameOver(a, b):
#a and b represent scores for a
#racquetball game
#Returns True if the game is over, False otherwise.
        return a==15 or b==15
```