ISE 314X
Computer Programing for Engineers

Chapter 9
Simulation and Design

Yong Wang
Assistant Professor
Systems Science & Industrial Engineering
Binghamton University

Objectives

- To understand Monte Carlo simulation
- To understand how to design complex programs

A Simulation Problem

- Player A plays racquetball with Player B
- Player B is slightly better than Player A
- Player B usually wins the matches

A Simulation Problem

- Shouldn't Player B, who is *a little* better, win *a little* more often?
- Simulate the game to see if slight differences in ability can cause such large differences in scores

Analysis and Specification

- Racquetball is played between two players
- One player starts the game by putting the ball in motion – *a serve*

Analysis and Specification

- Players try to alternately hit the ball to keep it in play – *a rally*
- The rally ends when one player fails to hit a legal shot

Analysis and Specification

- If the server wins the rally, a point is awarded to the server
- If the server loses, the other player will serve next rally and no point is awarded
- The first player to reach 15 points wins the *game*

Analysis and Specification

- The *skill level* is represented by the probability that the server wins the rally
- If a player's skill level is 0.60, it means the server wins a rally (and a point) with a probability of 60%

Analysis and Specification

- The program will
  - prompt the user to enter the skill levels of the players
  - simulate the play of multiple games
  - print a summary of the results

Analysis and Specification

- All inputs are assumed to be legal numeric values, no error or validity checking is required
- In each simulated game, player A serves first

Monte Carlo Simulation

- This type of simulation is called *Monte Carlo simulation*
- Named after Monte Carlo Casino in Monaco
- The results depend on chances/probabilities

PseudoRandom Numbers

- A random number generator is needed
- Python library `random` contains such functions

PseudoRandom Numbers

- Two functions of greatest interest are `randrange` and `random`
- Pseudorandom numbers generated by these functions are correlated with the computer's date and time
- Each time a program is run, a different sequence of random numbers is produced

PseudoRandom Numbers

- `randrange(1,6)` returns a random int number from `[1,2,3,4,5]`
- `randrange(10,20,2)` returns a multiple of 2 between 10 and 20 (including 10 but excluding 20)

PseudoRandom Numbers

```
>>> from random import randrange
>>> randrange(1,6)
5
>>> randrange(1,6)
1
>>> randrange(1,6)
3
>>> randrange(1,6)
2
>>> randrange(1,6)
5
>>> randrange(1,6)
4
```

PseudoRandom Numbers

- `random` generate pseudorandom float numbers uniformly distributed between 0 and 1 (including 0 but excluding 1)

PseudoRandom Numbers

```
>>> from random import random
>>> random()
0.7943280091289816
>>> random()
0.00049858619405451776
>>> random()
0.134123140081687
>>> random()
0.9872455453536165З
>>> random()
0.21429424175032197
>>> random()
0.72918328843408919
```

PseudoRandom Numbers

- Suppose Player A's winning probability is 70%, for each serve
- We generate a `random` num between 0 and 1
- There is a probability of 70% that the random number will be < 0.70, and the other 30% it will be ≥ 0.70

PseudoRandom Numbers

```
r = random()
if r < 0.70:
    scoreA = scoreA + 1
```

PseudoRandom Numbers

- If we use a variable `prob` to represent the probability (e.g., 70%) of winning the serve

```
if random() < prob:
    scoreA = scoreA + 1
```

Top-Down Design

- In the *top-down design*, a complex problem is divided into a set of smaller, simpler problems
- Each smaller problem is then divided into even smaller problems
- The little pieces are then put back together as a solution to the original problem

Top-Level Design

- The top-level design of the algorithm for the racquetball simulation

```
Print an introduction
Get the inputs: probA, probB, n
Simulate n games using probA and probB
Print a report on the wins for both players
```

Top-Level Design

- Print an introduction

```
def main():
    printIntro()
```

- Assume that there's a `printIntro` function that prints the instructions

Top-Level Design

- Get the inputs

```
def main():
    printIntro()
    probA, probB, n = getInputs()
```

- Assume there's already a function called `getInputs`

Top-Level Design

- Simulate *n* games of racquetball

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

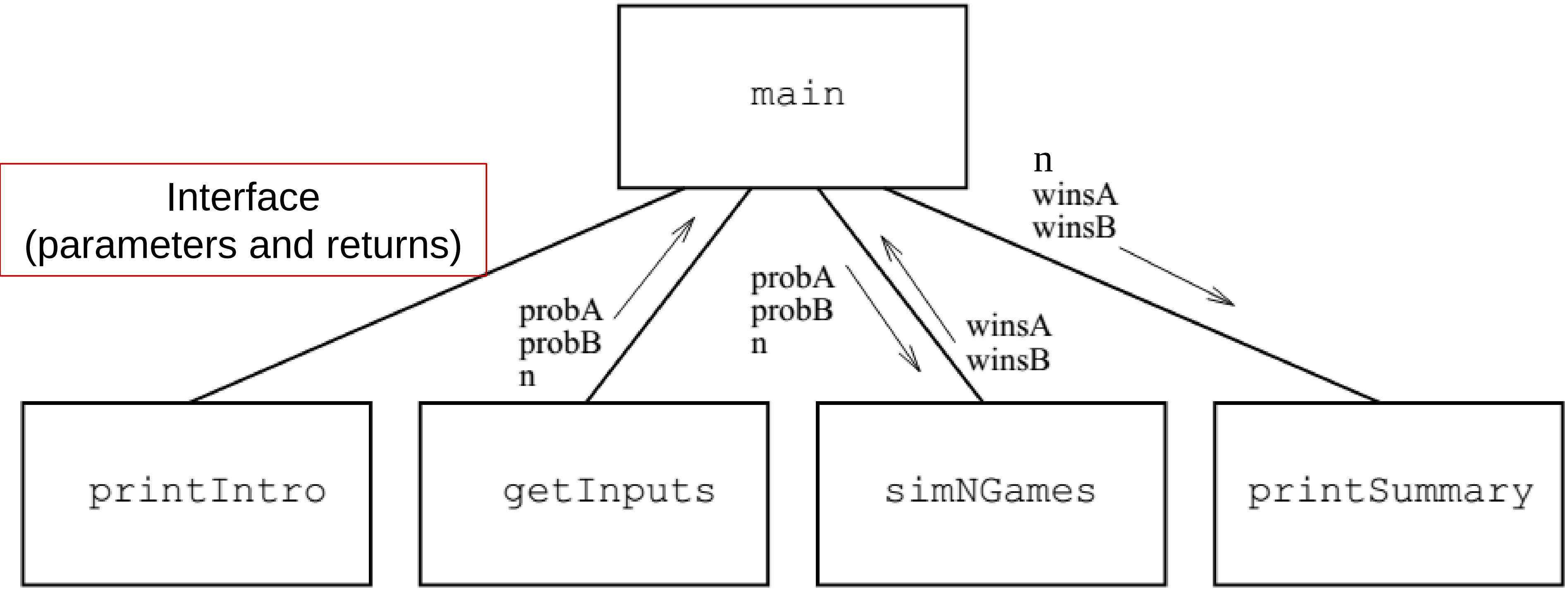
- Assume there is a function called `simNGames`

Top-Level Design

- Print the summary

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

Structure Chart



Second-Level Design

- Repeat the process for each module defined in the previous step and revise the structure chart accordingly

Second-Level Design

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

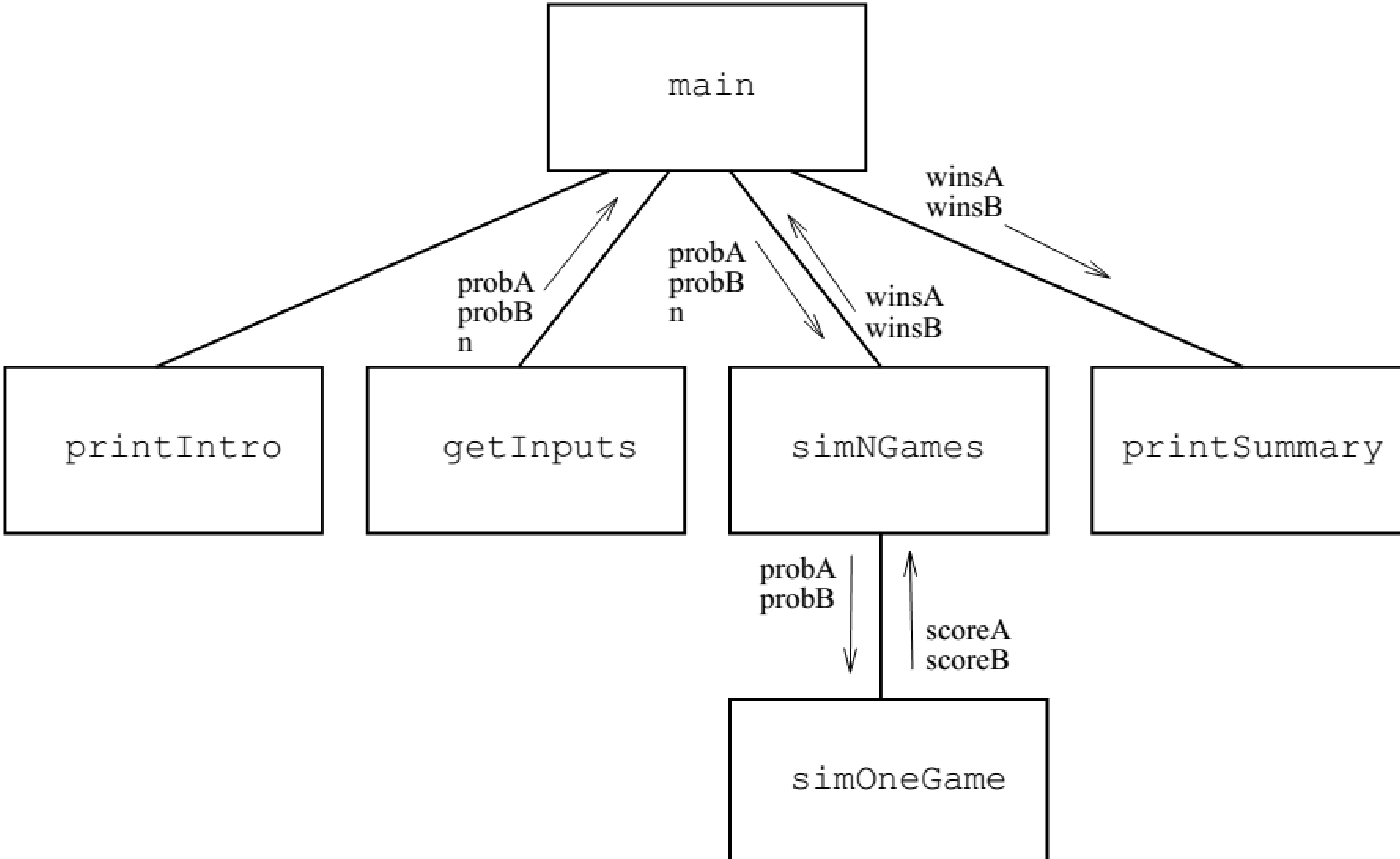
Second-Level Design

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

Designing simNGames

```python
def simNGames(n, probA, probB):
    '''
    Simulate n games and keeps track of how many wins
    there are for each player
    '''
    winsA = 0
    winsB = 0
    for i in range(n):
        scoreA, scoreB = simOneGame(probA, probB)
        if scoreA > scoreB:
            winsA = winsA + 1
        else:
            winsB = winsB + 1
    return winsA, winsB
```

Designing simNGames

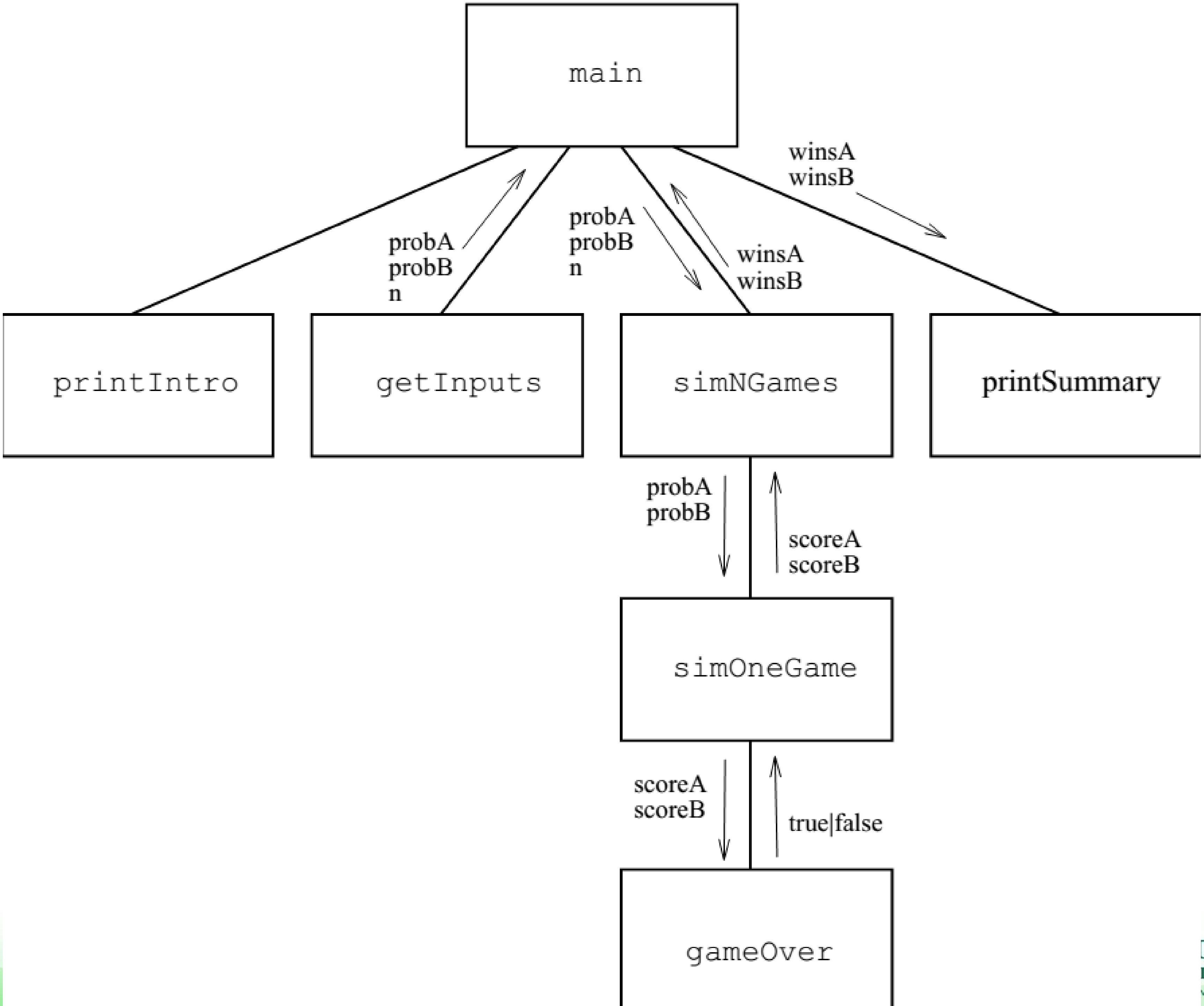


Third-Level Design

- In `simOneGame`, players keep doing rallies until the game is over

Third-Level Design

```
def simOneGame(probA, probB):
    '''
    Simulates a single game. Returns scores for A and B.
    '''
    serving = "A"
    scoreA = 0
    scoreB = 0
    while not gameOver(scoreA, scoreB):
        if serving == "A":
            if random() < probA:
                scoreA = scoreA + 1
            else:
                serving = "B"
        else:
            if random() < probB:
                scoreB = scoreB + 1
            else:
                serving = "A"
    return scoreA, scoreB
```


main
winsA
winsB
probA
probB
n
probA
probB
n
winsA
winsB
printIntro
getInputs
simNGames
printSummary
probA
probB
scoreA
scoreB
simOneGame
scoreA
scoreB
true|false
gameOver

Finishing Up

```
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
```

Finishing Up

- Print the summary

```
def printSummary(n, winsA, winsB):
    '''Prints a summary of wins for each player.'''
    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))
```

Finishing Up

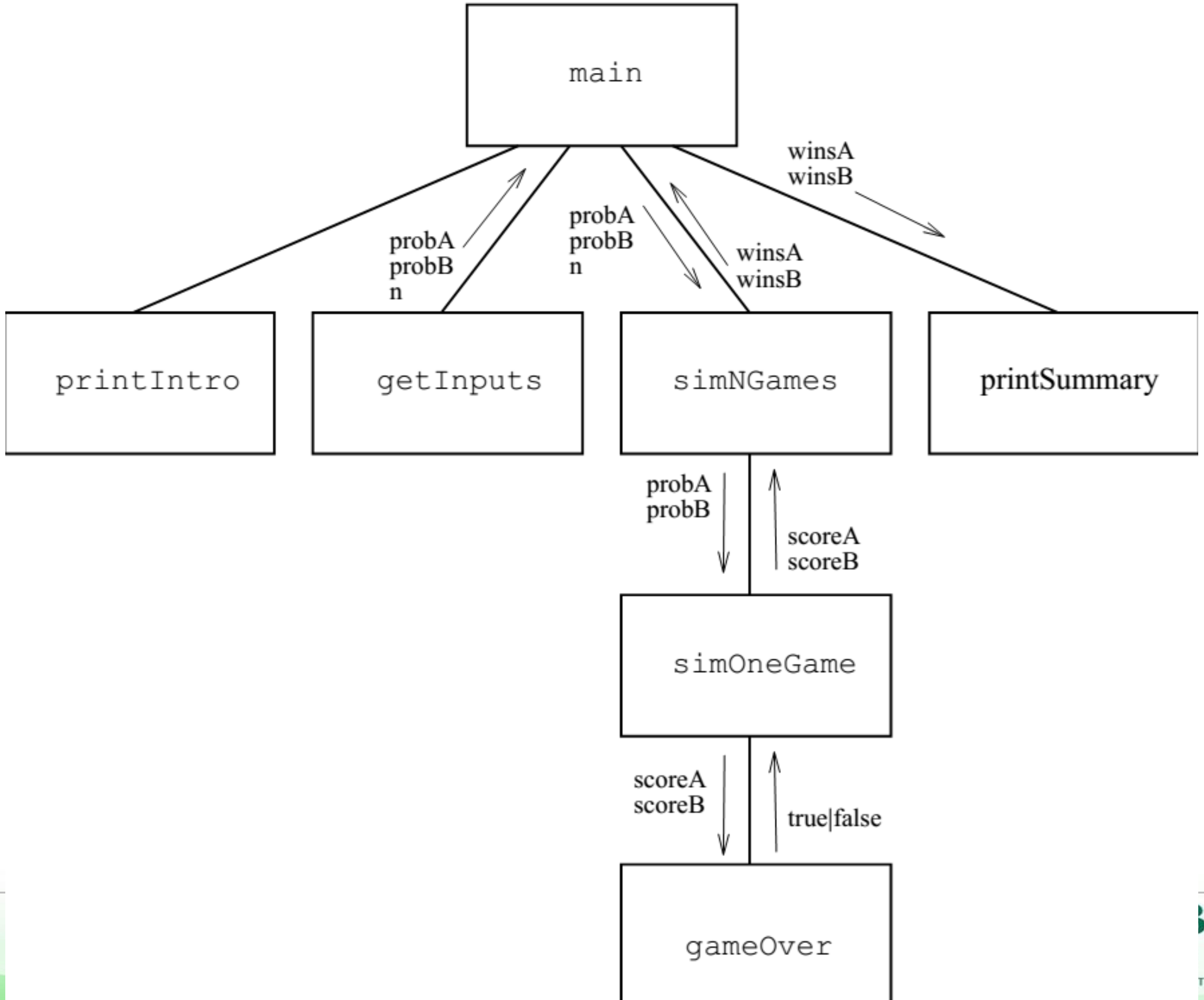
- It can be called as a stand-alone program as well as imported as a library

```
if __name__ == '__main__':
    main()
```

- The program is `rbgame.py`

Unit Testing

- Systematically test a program: start at the lowest levels and test each component



Unit Testing

```
>>> import rbgame
>>> rbgame.gameOver(0,0)
False
>>> rbgame.gameOver(5,10)
False
>>> rbgame.gameOver(15,3)
True
>>> rbgame.gameOver(13,15)
True
```

Unit Testing

```
>>> rbgame.simOneGame(0.5, 0.5)
(11, 15)
>>> rbgame.simOneGame(0.5, 0.5)
(15, 13)
>>> rbgame.simOneGame(0.4, 0.9)
(1, 15)
>>> rbgame.simOneGame(0.9, 0.4)
(15, 0)
>>> rbgame.simOneGame(0.4, 0.6)
(10, 15)
>>> rbgame.simOneGame(0.4, 0.6)
(9, 15)
```

Simulation Results

- Is small differences in skills lead to large differences in final score?

Simulation Results

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

What's the prob. player A wins a serve?0.6
What's the prob. player B wins a serve?0.65
How many games to simulate?10000

Games simulated: 10000
Wins for A: 4012 (40.1%)
Wins for B: 5988 (59.9%)
```

Simulation Results

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

What's the prob. player A wins a serve?0.65
What's the prob. player B wins a serve?0.6
How many games to simulate?10000

Games simulated: 10000
Wins for A: 6759 (67.6%)
Wins for B: 3241 (32.4%)
```

Simulation Results

- **Why do we have the difference?**