Lecture 18

Topics: *Chapter 9. Simulation and Design*

- 9.2 Pseudo-random numbers
- 9.1 Simulating Racquetball
- 9.3 Top-Down Design
- 9.4 Bottom-Up Implementation

9.2 Pseudo-random numbers

There is nothing random about computers, they are instruction-following machines.

So the numbers that we get with the help of computers are called pseudo-random numbers.

9.2 Pseudo-random numbers

There is nothing random about computers, they are instruction-following machines.

So the numbers that we get with the help of computers are called pseudo-random numbers.

The idea of the *pseudo-random number generator*:

- start with some *seed* value,
- feed into a function to produce a «random» number,
- next time the random number is needed, the current value is fed back into the function to produce a new number.

The function has to be carefully chosen.

9.2 Pseudo-random numbers

There is nothing random about computers, they are instruction-following machines.

So the numbers that we get with the help of computers are called pseudo-random numbers.

Python provides a library module `random` that contains a number of useful functions for generating pseudo-random numbers.
The functions in this module derive an initial seed value from the date and time when the module is loaded, so each time the program is run we get a different seed value.

9.2 Pseudo-random numbers

Random module

Few functions of interest to us:

`randrange([start], stop[, step])`
– returns a randomly selected element from `range(start, stop, step)`; does not actually build a range object

`randint(a,b)`
– returns a random integer `N` such that $a \leq N \leq b$.
alias for `randrange(a, b+1)`.

`random()`
– returns the next random floating point number in the range [0.0, 1.0)

9.2 Pseudo-random numbers

Random module

Few functions of interest to us:

`choice(seq)`
– returns a random element from the non-empty sequence seq.
If `seq` is empty, raises `IndexError`.

`shuffle(x[, random])`
- shuffles the sequence `x` in place.
The optional argument random is a 0-argument function returning a random float in [0.0, 1.0); by default, this is the function random().

More can be found here:
https://docs.python.org/3.1/library/random.html

9.1 Simulating Racquetball

We have almost all tools to solve interesting problems, i.e. the ones that are difficult or impossible to solve without the ability to write and implement computer algorithms.

Simulation – is one of techniques for solving real-world problems

Examples of problems that are solved with computer simulation:

- weather prediction,
- aircraft design,
- video games,
- etc.

9.1 Simulating Racquetball

Let's develop/write a program that will be a simple simulation of the racquetball game.

Racquetball is a sport played between two players using racquets to strike a ball in a four-walled court.
To start the game, one of the players puts the ball into play (called serving). The players then alternate hitting the ball to keep it in play. This is a rally. The rally ends when one of the players fails to hit a legal shot. The player who misses the shot loses the rally. If the loser is the player who served, then service passes to the other player. If the server wins the rally, a point is awarded. Players can only score points during their own service.
The first player to reach 15 points wins the game.

Two links to video demonstrations:

https://youtu.be/A-uHQxcxVa0
https://youtu.be/WveNMboopls

9.1 Simulating Racquetball

Simulation of the racquetball game

Summary:
two players,
four-walled court,
serving and receiving,
the rally ends when one of the players fails to hit a legal shot,
the player who misses the shot loses the rally,
if the loser is the player who served, then service passes to the other player,
if the server wins the rally, a point is awarded,
players can only score points during their own service,
the first player to reach 15 points wins the game.

9.1 Simulating Racquetball

Simulation of the racquetball game

Why do we want to simulate this play? What is of interest for us?

- the correspondence between ability-level of the players and the number of wins.

Is it true that if one of the players plays *slightly better* than the other, he/she should win *slightly more* games?

9.1 Simulating Racquetball

Simulation of the racquetball game

Why do we want to simulate this play? What is of interest for us?

- the correspondence between ability-level of the players and the number of wins.

Is it true that if one of the players plays *slightly better* than the other, he/she should win *slightly more* games?

Convention: the ability-level of players will be represented by the probability that the player wins the rally when he or she serves.

P_A – the probability that the player A wins on his/her serve

P_B – the probability that the player B wins on his/her serve

9.1 Simulating Racquetball

Simulation of the racquetball game

Specification:

Input:

the service probabilities of two players
the number of games to be simulated

Output:

the number of games simulated
the number of wins for Player A
(and what percent of the total number of games is it)
the number of wins for Player B
(and what percent of the total number of games is it)

Assumption:

in each game player A serves first

9.3 Top-Down Design

Idea:

- start with the general problem and try to express a solution in terms of smaller problems, i.e. break the original problem into sub-problems;

- apply the same technique to each smaller sub-problem;

- eventually the problems get so small that they are trivial to solve.

9.3 Top-Down Design

Simulation of the racquetball game

print an introduction
get Pa and Pb
get n (the number of games)
simulate n games using Pa and Pb
print the report on the wins for Player A and Player B

9.3 Top-Down Design

Simulation of the racquetball game

print an introduction easy (code it!) Intro()
get Pa and Pb
get n (the number of games)
simulate n games using Pa and Pb
print the report on the wins for Player A and Player B

9.3 Top-Down Design

Simulation of the racquetball game

print an introduction	easy (code it!)	Intro()
get Pa and Pb	easy (code it!)	getInput() returns
get n (the number of games)	easy (code it!)	Pa, Pb, n

simulate n games using Pa and Pb

print the report on the wins for Player A and Player B

9.3 Top-Down Design

Simulation of the racquetball game

print an introduction — easy (code it!) — Intro()

get Pa and Pb — easy (code it!)

get n (the number of games) — easy (code it!)

getInput() returns Pa, Pb, n

simulate n games using Pa and Pb break into smaller sub-problems

simNgames(n,Pa,Pb) returns WinsA and WinsB

print the report on the wins for Player A and Player B

9.3 Top-Down Design

Simulation of the racquetball game

print an introduction — easy (code it!) — Intro()

get Pa and Pb — easy (code it!)

get n (the number of games) — easy (code it!)

getInput() returns Pa, Pb, n

simulate n games using Pa and Pb — break into smaller sub-problems

simNgames(n,Pa,Pb) returns WinsA and WinsB

simOneGame(Pa,Pb) returns 1 if Player A wins, and 0 otherwise — easy (code it!)

print the report on the wins for Player A and Player B

9.3 Top-Down Design

Simulation of the racquetball game

print an introduction — easy (code it!) — Intro()

get Pa and Pb — easy (code it!)

get n (the number of games) — easy (code it!)

getInput() returns Pa, Pb, n

simulate n games using Pa and Pb — break into smaller sub-problems

simNgames(n,Pa,Pb) returns WinsA and WinsB

simOneGame(Pa,Pb) returns 1 if Player A wins, and 0 otherwise — easy (code it!)

print the report on the wins for Player A and Player B — easy (code it!) — report(n,wA,wB)

9.3 Top-Down Design

Simulation of the racquetball game

```
def main():
    Intro() # print an introduction

    Pa, Pb, n = getInput() # get input from the user

   # simulate n games,
   # return the result of simulation
    WinsA, WinsB = simNgames(n,Pa,Pb)

    report(n,WinsA,WinsB) # report back to the user
```

We outlined the sub-parts of the main function and the relationships between them (what is returned by each function, what is supplied as an argument to each function, which function goes after which)

9.3 Top-Down Design

Simulation of the racquetball game

```
def main():
    Intro() # print an introduction

    Pa, Pb, n = getInput() # get input from the user

   # simulate n games,
   # return the result of simulation
    WinsA, WinsB = simNgames(n,Pa,Pb)

    report(n,WinsA,WinsB) # report back to the user
```

At each level of design we need to determine important characteristics of something and ignore other details (called abstraction)

9.3 Top-Down Design

Simulation of the racquetball game

We implement three methods:

```
Intro()
GetInput()
report(n,wA,wB)
```

see program racquetball.py

9.3 Top-Down Design

simOneGame(Pa,Pb)

- returns 1 if Player A wins, and 0 otherwise

- players keep doing rallies until somebody collects 15 ponts: this suggests an **infinite loop**

- we need to keep the track of players' scores (`scoreA`, `scoreB`)

- How to deal with probabilities? Where to apply them?

9.3 Top-Down Design

simOneGame(Pa,Pb)

- returns 1 if Player A wins, and 0 otherwise

- players keep doing rallies until somebody collects 15 ponts: this suggests an **infinite loop**

- we need to keep the track of players' scores (`scoreA`, `scoreB`)

- How to deal with probabilities? Where to apply them?

```
set serving to Player A
scoreA = 0
scoreB = 0
while scoreA != 15 and scoreB != 15
   simulate one serving
   update the status of the game
return who wins(1 or 0)
```

9.3 Top-Down Design

simOneGame(Pa,Pb) - Probabilities

Let's talk about probabilities:

what does it mean to have a probability of 70%?

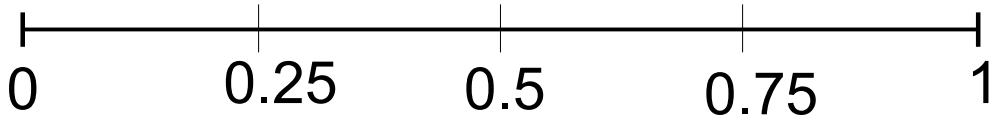
9.3 Top-Down Design

simOneGame(Pa,Pb) - Probabilities

Let's talk about probabilities:

what does it mean to have a probability of 70%?

Suppose we generate a number between 0 and 1.



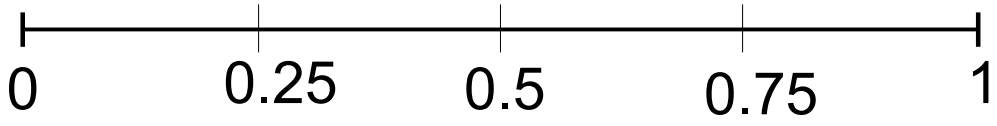
9.3 Top-Down Design

simOneGame(Pa,Pb) - Probabilities

Let's talk about probabilities:

what does it mean to have a probability of 70%?

Suppose we generate a number between 0 and 1.



Exactly 70% of the interval 0 ... 1 is to the left of 0.7.

So 70% of time the random number will be < 0.7, and it will be ≥ 0.7 in other 30% of the time.

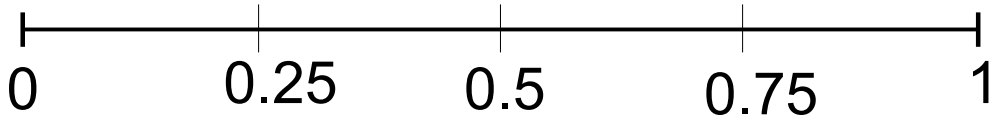
9.3 Top-Down Design

simOneGame(Pa,Pb) - Probabilities

Let's talk about probabilities:

what does it mean to have a probability of 70%?

Suppose we generate a number between 0 and 1.



Exactly 70% of the interval 0 ... 1 is to the left of 0.7.

So 70% of time the random number will be < 0.7, and it will be ≥ 0.7 in other 30% of the time.

Therefore let's use the following piece of code

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

9.3 Top-Down Design

simOneGame(Pa,Pb) - Probabilities

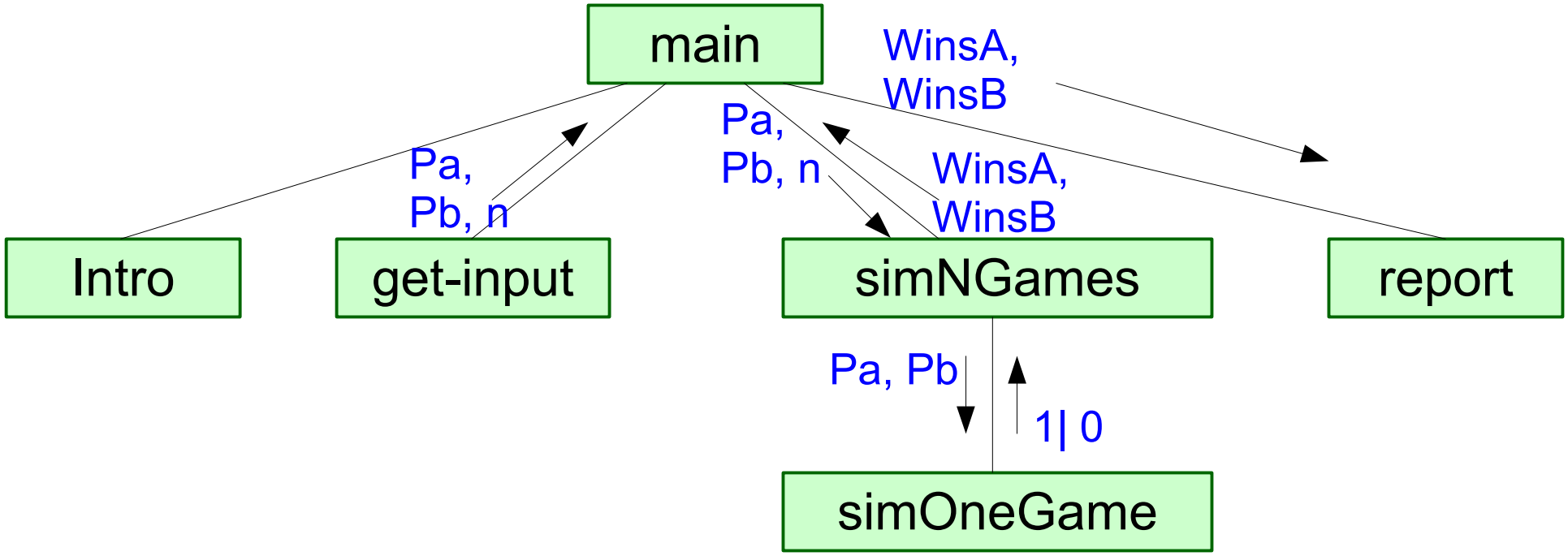
We proceed to the implementation of `simOneGame`

Followed by the implementation of simNGames

see program racquetball.py

9.3 Top-Down Design

Simulation of the racquetball game



structure chart

9.4 Bottom-Up Implementation

Simulation of the racquetball game

Did you notice that our implementation started with “easy” sub-sub-....-sub parts?

- this is called *bottom-up implementation*

Each implemented sub-sub...-sub part needs thorough testing, called *unit testing*