Simulation and Design

Import random as rnd

Top-down design

Bottom-up design

Prototyping and spiral development

Simulation

Simulation can solve real-world problems by modelling real-world process to provide otherwise unobtainable information

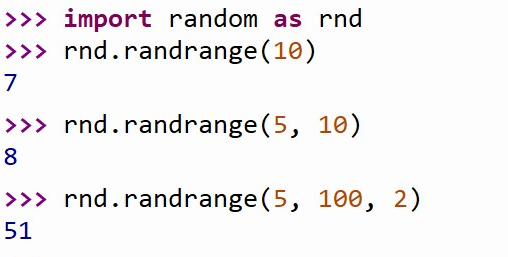
pseudorandom numbers

a pseudorandom number generator works by starting with a seed value – current date and time. This value is given to a function to produce a “random” number.

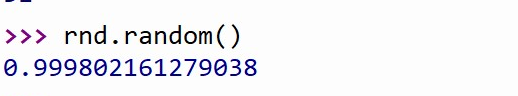
The next time a random number is required, the current value is fed back into the function to produce a new number.

This sequence of numbers appears to be random, but if you start the process over again with the same seed number, you’ll get the same sequence of “random” numbers.

The random-range function would provide a random number between 0 and 9 (the last value 10 not included as usual)



The random function would generate a random value between 0 and 1 (not include 1, it can be 0)

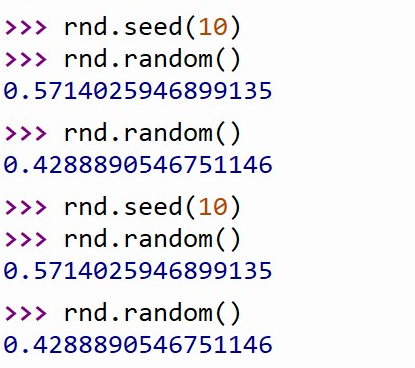


Regenerate the sequence



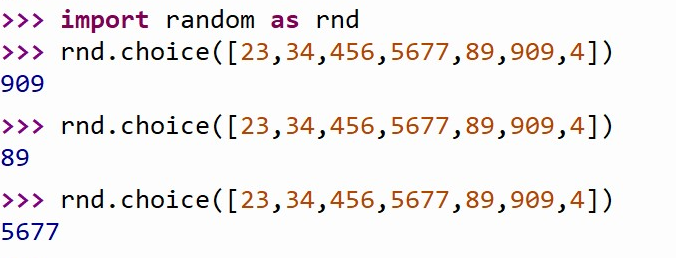
The seed function assigns the random number generator a fixed starting point (to five reproducible behaviour during testing)

It is fine to give any number inside the bracket, as long as they keep the same.

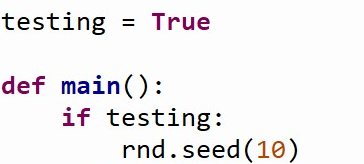




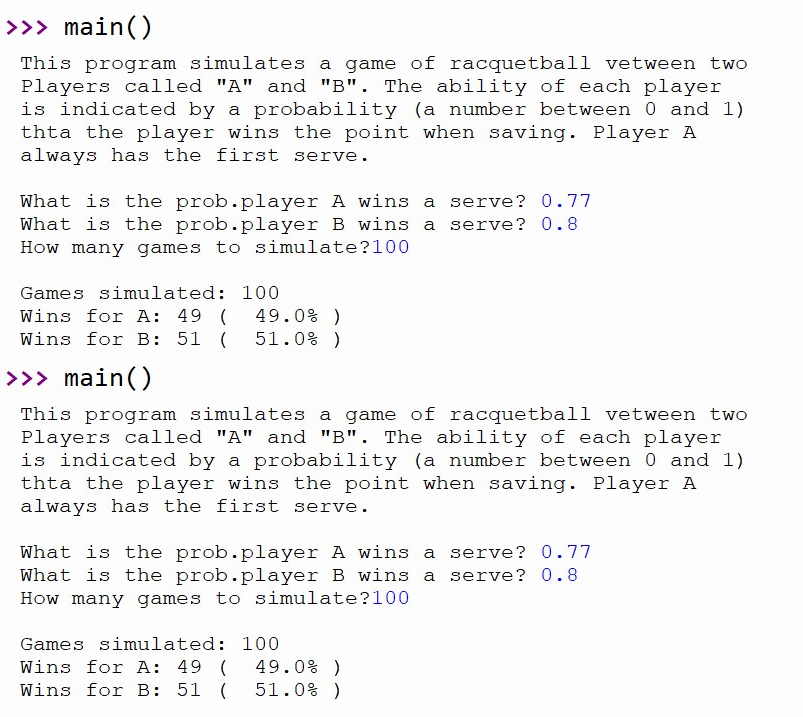
Choice () -> choose a random member of a list



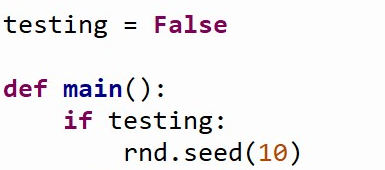
About the seed function

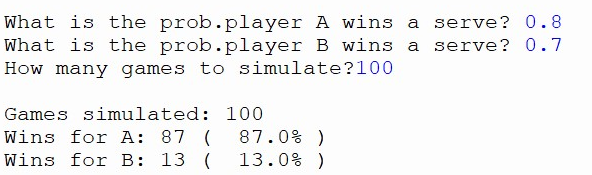


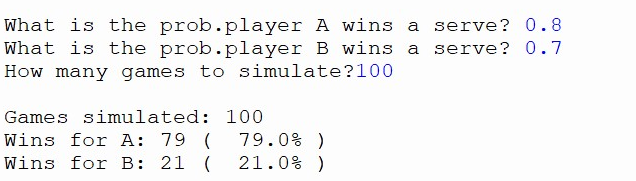
Now it will return the same answer once you input the same possibility of A and B and the times of the games.



But if you turn testing into false, it would not be the same







Top-down design

In a top-down design, a complex problem is expressed as a solution in terms of smaller, simpler problems. It just keeps dividing the size of the problems until the problems are trivial to solve. The smaller pieces are then put back together as a solution to the original problems.

(input, process, output)

1. Run the introduction of the problem
2. Get the input
3. Simulate the entire process
4. Explain the research

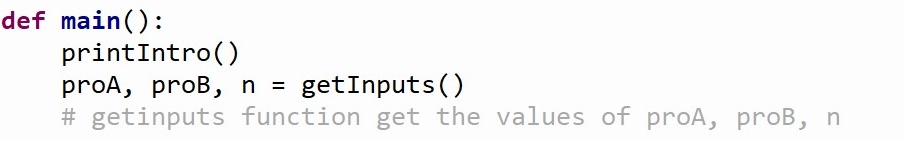
Assume that all the components needed to implement the algorithm have been written already, and that your task is to finish this top-level algorithm using those components.

First, we print the introduction

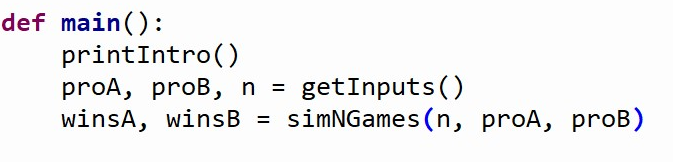


The next step is to get the input

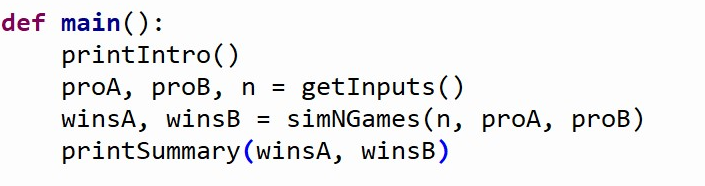
Let’s assume there’s already a component that can do that called get-Inputs ().



For the output, what values would you need to get back?



Finally, print the conclusion



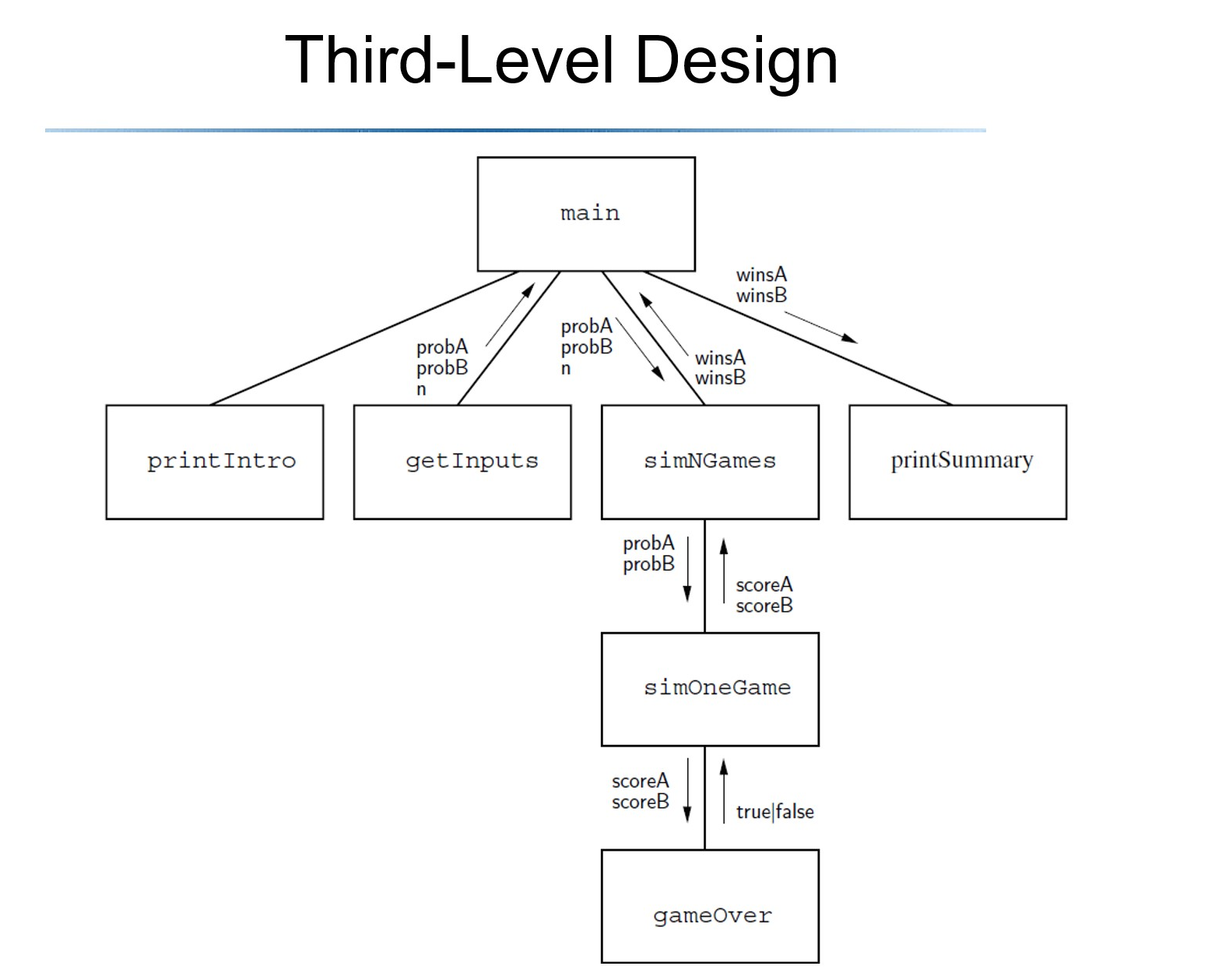
Main () in the programming projects was an API (interface)

P: the name, parameter, and expected return values of these functions have been specified. This information is known as the interface (API) or signature of the function.

Separation of concerns

Having this information (the signatures), allows us to work on each of thses pieces independently.

From top to bottom!!!



Summary of top-down design process

1. We start at the highest level of our structure chart and worked our way down
2. At each level, we began with a general algorithm and refined it into precise code
3. This process is sometimes referred to as step-wise refinement.

Summary of the design process:

1. Express the algorithm as a series of smaller problems
2. Develop an interface for each of the small problems
3. Detail the algorithm by expressing it in terms of its interfaces with smaller problems
4. Repeat the process for each smaller problem

Bottom-up implementation

Even though we have been careful with the design, there’s no guarantee we haven’t introduced some silly errors.

Implementation is best done in small pieces. (start with the functions you know you need to put together)

Unit testing

Testing one element at a time

Once unit testing is done, you can take a step can check the entire program.

Start at the lowest levels of the structure, testing each component as it’s completed.

When testing, need to have reproducible behaviour. That is, the program behaves the same way each time it is executed.

Testing each component in the whole program in this manner is called unit testing.

Then need end-to-end, or integration(结合), testing

Prototyping and spiral development

(basis of the agile design methodologies)

Another approach to program development is to start with a simple version of a program, and then gradually add features until it meets the full specification.

The initial stripped-down version is called prototype. (the method is sometimes called rapid prototyping)

Prototyping often leads to a spiral development process.

We take many mini-cycles through the development process as the prototype is incrementally expanded into the final programs.   
(at each step, consult with the client)

Spiral development means that you are going to recur again and again, solve it, test it, and go back to the clients and ask the if it’s ok, and keep on doing these processes.

Obviously, you would never have a big picture of what would be overall product and you can not design from the very firstly.

The art of design

Spiral development is not an alternative to top-down design as much as a complement to it – when designing the prototype you’ll still be using top-down techniques.

Good design is as much creative process as science, and as much, there are no hard and fast rules

We combine top-down into the spiral model, whenever we get actually a small development or small task from the client, we use top-down approach to solve that problems.

And then go back to the client, and then when we get the increment functionality, we have done again go ahead and keep on doing the repeated the task.

Normally it prefers to go down no more than 20 lines for each function, the complexity level of function would be increase and it would cause problems if the lines of the function keep increasing.

So you need to do unit testing to check the function, make sure it is working well and then move on.