# Defining solutions

## Introduction

Computer science is all about solving problems. The invention of the computer has given us an unprecedented ability to conceptualize and overcome problems, real and imaginary. Before, you'd have to drop a plane from the sky to test if it will fly. Or gather weather data and make predictions based on patterns. Now it is possible to model scenarios realistically and create a solution virtually before sending up a weather balloon or attaching even one wing to an airplane. This reading is about how to define solutions to problem statements by articulating the problem, formulating a model and finetuning the solution.

## Problem statement

The first step to solving a problem is to articulate it. I need to achieve A, with tools T, given constraints C. By way of fleshing out the notion of good practices when engaging with designing a solution, let's consider a real-world problem. Say you have to design an app that will entertain a child by playing a guessing game. On the face of it, this seems like a straightforward task: computer + game = happy child. But, before you start, it is worth it to determine some details.

1. How will the child interact with the computer?
2. What level of questions should be asked?
3. From where are these questions generated?
4. How will they be stored?
5. How will the answers be checked?
6. How will the program start and end?

In this scenario, the issues are both hardware and software related. First, is the child of an age where safety measures are not required? In other words, it is important that the ideal solution is contemplated with consideration to how the final application is to be used. Giving a toddler an expensive laptop might buy a morning of peace, but then it might cost a month’s wages! This same thinking must be extended to the application that is visualized. Will your player be engaging with your application with a strong internet signal? Are there other constraints like the output needing to be compatible with different types of mobile phones? Is there a specific operating system or browser that must be catered to? These are all questions you need to consider before you start creating the app.

## Formulate a model

Now that there is an idea of some of the constraints on the project, potential solutions can be proposed. Let's imagine the child is of a computer-friendly age, the application need only be run on a laptop, all storage will be done locally (so there are no internet connectivity issues) and the input can be done using a keyboard on the command line. Great, now it is possible to further explore the project.

Having established a scope for the project it is time to consider what it is the computer is required to do. An algorithm can be defined as a precise sequence of instructions to solve a problem. It can be helpful to first generalize the problem and then more precisely outline the sequence of required instructions before coding an implantation. This is like taking an algorithmic approach to solving the issue. A programmer will often gain an intuition on what the solution should look like by first sketching out the problem using pseudocode. This can be a text-based description that details the requirements.

```
Generate a question
Take in user input
Compare input with answer
```

```
Return a result.
```

Plotting out the steps in a list is a good first step. Now, consideration can be placed on how to generate a question that is age appropriate. Dynamically generating questions from an online encyclopedia or some other online source might be time-consuming. Alternatively, you could identify a source of already compiled questions and answers.

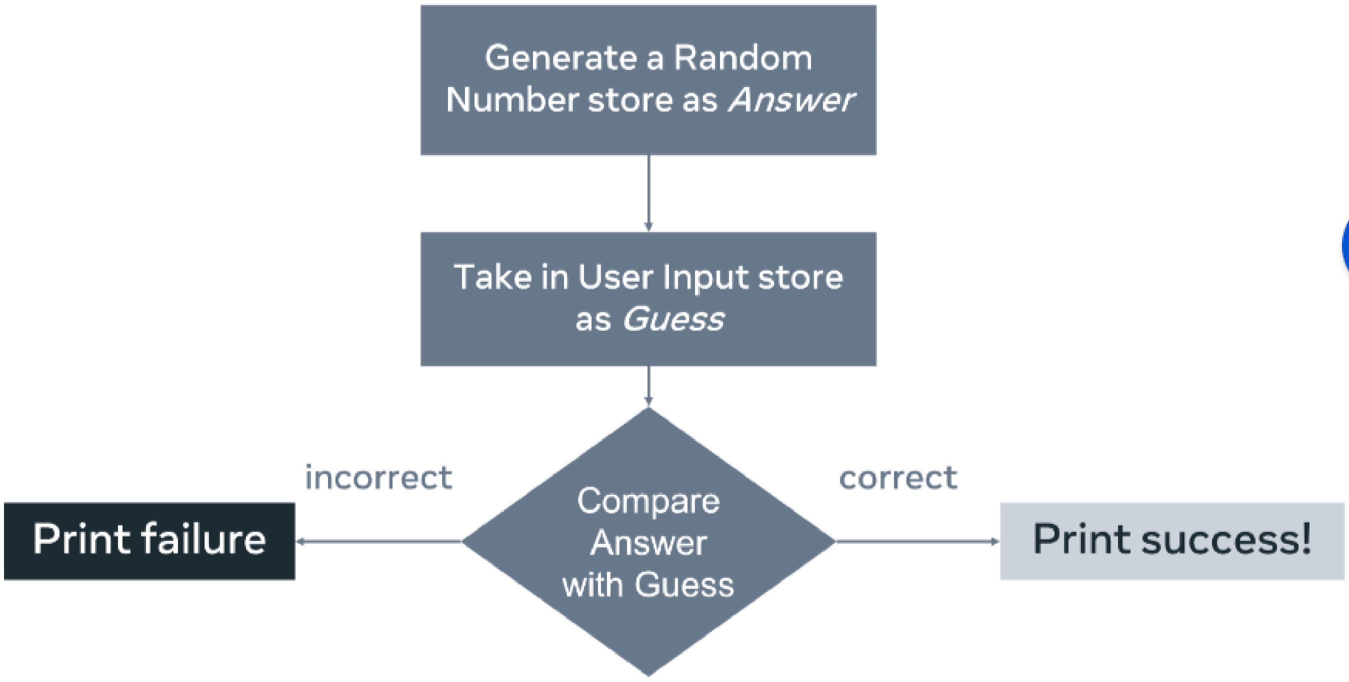
Second, how is user input taken? It has been established that the keyboard is viable, but how capable is the child at typing? Comparing a solution with text offers its own challenges. Would it need to be a direct match, would spelling, punctuation and capitalization be considered? Potentially, the questions could be multiple-choice. Raising awareness of all of these considerations is the purpose of outlining the problem statement clearly rather than charging into a solution.

### Finetuning the solution

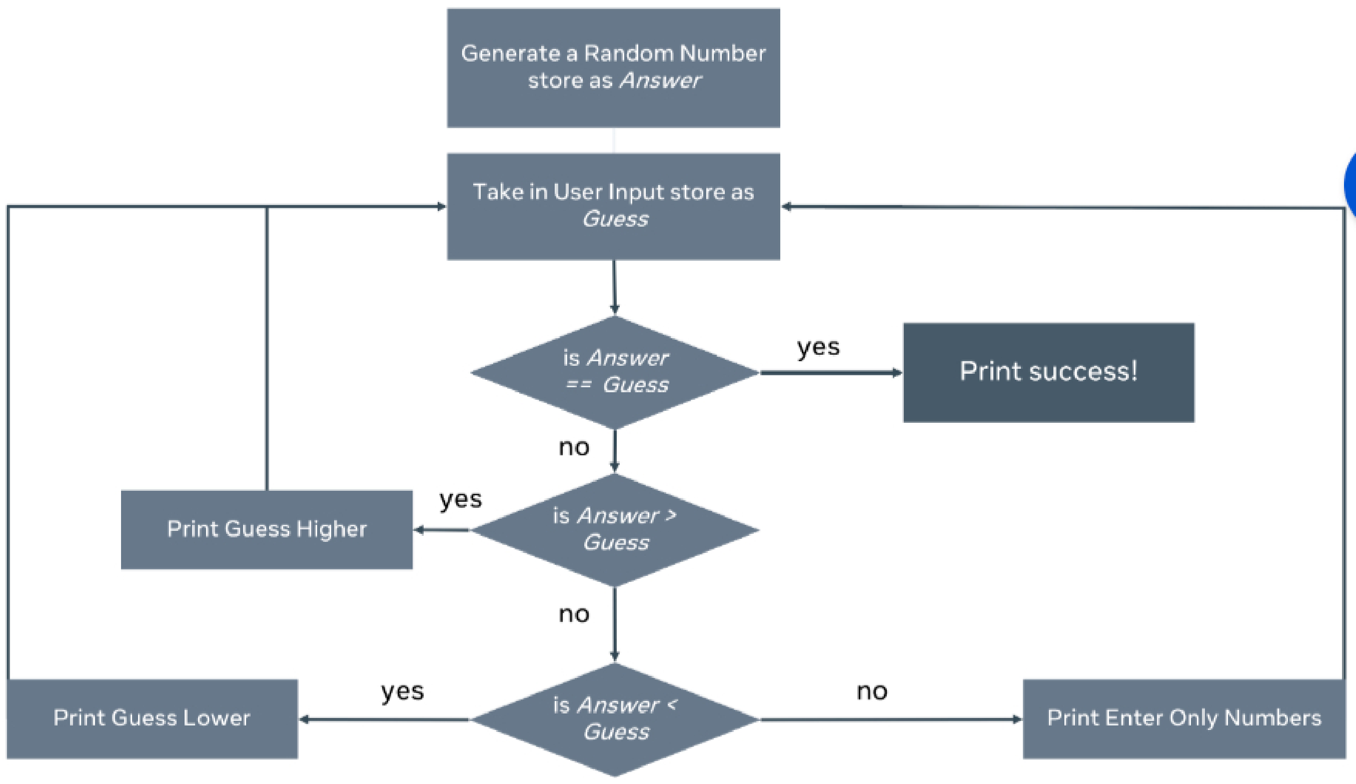
On reflection, you determine that a trivial knowledge approach will offer too many obstacles so the project is further defined as a number-guessing game. It's appropriate for all ages and there's no need to use external sources. And it should be suitable to implement with most programming languages.

```
Generate a random number
Take in user input store in a variable
Compare input with answer
Return a result.
```

Having outlined the general requirements, it is possible to fill in the finer points. Below is a flow chart of the pseudocode and it can give further insights into how to best frame the solution.


Generate a Random Number store as *Answer*
Take in User Input store as *Guess*
incorrect
correct
Print failure
Compare Answer with Guess
Print success!

Examining the flowchart identifies further considerations. Will the program run only once? What can be added to enhance the user experience? Printing out a failure message might not be a great output. Instead, an option might be to offer another chance to guess. Further considerations might be to provide prompts that could be used to steer the user toward the correct answer. And, you should decide what happens if the user enters a non-number into the program.



Now that the various conditions have been outlined it's possible to pick a language and implement a solution. By taking a systematic approach to problem-solving you identified some potential issues before the project began and you could add some course corrections. Upon evaluating the solution the project might now change scope. In place of creating a game for a child to implement, the program might instead establish an environment so that the child can create the game themselves, using the clear-cut descriptions as a guide.

## Conclusion

Computers are a great way to model programs and a means of implementing a solution. It's the programmer's responsibility to employ a systematic approach in developing solutions. The same procedure used to identify a good project for a child can be applied to creating an application for work. It's imperative to establish these good working habits early in your career and to maintain them as your coding proficiency progresses.