# Computational thinking as habits of mind for mathematical modelling

*Keng-Cheng Ang*
kengcheng.ang@nie.edu.sg

National Institute of Education
Nanyang Technological University
Singapore

**Abstract:**

*The growing interest in computational thinking and its use in problem solving had led teachers and educators, as well as other researchers, to ponder over what it means and how best to introduce such a notion to students in schools. Many ideas on "teaching computational thinking" have also been suggested, and in many countries, courses on coding or computer programming have been made very popular as more people begin to believe that the ability to write code is an important skill in this increasingly digital world. In this paper, we focus on the habits of mind that are related to computational thinking and that can be developed from learning to code. Some of these habits include looking at trends in data and analyzing them, examining a process and simulating it, and systematically constructing a solution to a problem. More specifically, we shall discuss how these habits of mind can enhance and support one's skills and competencies in the context of mathematical modelling, using three examples. Individually, each example illustrates some aspects of computational thinking applied to the modelling tasks. Collectively, through these examples, we attempt to demonstrate that the related habits of mind of computational thinking, developed through computer programming exercises, could strengthen one's ability and expand one's capability of tackling modelling tasks in a significant, albeit sometimes subtle way.*

## Introduction

In recent years, the notion of computational thinking and its impact and implications in education have been the subject of much discussion and debate among researchers. This is in part, brought about by the viewpoint piece on computational thinking presented by Wing [14] and in part a natural consequence of the increasingly digital world that we live in. Much of the academic discussions surrounding this topic had focused on the concept or definition of computational thinking, which Wing unfortunately, or perhaps, intentionally did not provide, how it is related to problem solving and how it can be taught.

Nonetheless, most would agree on the different aspects of computational thinking, which would include constructs and notions like abstraction, creating algorithms, decomposition, pattern recognition and so on. In recent years, many educators and researchers have published articles and papers on the topic, explaining what these constructs mean, and how they appear in problem solving [8, 11, 13, 15]. Many have also recommended ways of "teaching" computational thinking, while still others have advocated that it should be included in different subjects and be made a significant part of the school curriculum [6, 12].

Wing argued that is it important that every student is taught "how a computer scientist thinks". To some, this may be interpreted as to mean it is important to teach computer science to every student. Nardelli, however, suggested that it is more critical to stress the "educational value of informatics, that is, computer science, for all students", and proposed that we discuss what to teach and how to evaluate competences regarding informatics rather than teaching and evaluating computational thinking [9].

In the context of mathematics learning and teaching, an educator would have to grapple with the notion of computational thinking and its practice while balancing with the concept and thrust of mathematical thinking that has been the objective of teaching and learning mathematics. It is sometimes argued that computational thinking is very similar to mathematical thinking [5], and to a large extent, it is not unreasonable to see computational thinking as mathematical thinking with a powerful calculating tool.

Against the backdrop of these different perceptions of what computational thinking means, and the various suggestions and recommendations for including its teaching and practice in the school curriculum, as well as its relationship with mathematical thinking, a typical mathematics teacher is faced with many challenges. Yet, perhaps it is no longer useful or relevant to ask what exactly constitutes computational thinking, or imagine what can be done. Rather, it may be more important to think about what can be derived from learning and practicing computer programming or coding, and how that can be useful and relevant in problem solving, particularly in mathematics.

Specifically, it has been suggested that the practice of programming helps one develop some very useful *habits of mind* that will eventually support the practice of mathematical modelling [2]. The focus of this paper, therefore, is to explicate and illustrate this idea.

## Habits of mind

What exactly do we mean by habits of mind? As a very general view, the term refers to a set of dispositions that enables us to behave in an intelligent manner when faced with problems. Behavioral scientists have studied this aspect of human development for some time now, and some researchers have narrowed these dispositions down to "16 habits of mind" [3], with a disclaimer that the list is not complete. Across the world, many institutions have adopted these frameworks and designed programmes within school curricula to help school children develop these habits and apply them in problem solving.

In the present discussion, it is more perhaps useful to adopt a perspective proposed by Goldenberg [4], who had said that habits of mind are "*ways of thinking that one acquires so well, makes so natural, and incorporates so fully into one's repertoire, that they become mental habits – not only can one draw upon them easily, one is likely to do so*". In other words, habits of mind are not just dispositions that make one behave intelligently but are a set of internalized practices of critical thinking that the one could, and often would, employ in a problem situation.

Therefore, if computational thinking is about habits of mind, then it is necessary to first recognize and identify the dispositions associated with it. As Wing explained, to think computationally is to think like a computer scientist, and "thinking like a computer scientist means more than being able to program a computer". While learning Scratch or Python may be important, it is not the end goal; more important than these are the mental habits of the people who make use of computers, computing tools, computing constructs, computational methods and computational models to solve problems.

Nevertheless, just as calculation is the machinery of mathematics, computer programming is the machinery of computational thinking. Just as learning to calculate (or to solve mathematical problems) is a means to developing a mathematical mind, similarly, learning to code (or to write a program to build a computational model) is a means to developing habits of mind in computational thinking. More specifically, these habits and ways of thinking are particularly useful, and even essential in some cases, in solving mathematical modelling problems.

For instance, many modelling tasks or situations involve data, either given or collected, which can be used in some ways to construct a mathematical model. One needs to look for trends, or fit curves and functions, or find suitable parameter values. These often involve numerical or computational methods and looking for and recognizing some patterns in the data before devising appropriate strategies to use the data. Working with data is one aspect that many programmers have had to learn to do when learning to program.

Another example is in the development of simulation models. In order to successfully run a simulation of a situation or process, the programmer will need to understand the process very well. Sometimes, some form of abstraction needs to be done, and some assumptions made to keep the problem tractable. Often, an algorithm is written to flesh out the step by step procedure before the code is written. These are all good habits of mind that helps one be systematic and organized, and to clarify the process of constructing a simulation model.

Coding or programming exercises provide the platform for these habits to be developed. These exercises often require understanding of the methods and schemes used, such as in computational methods of solving a mathematical problem or in implementing complex algorithms. In order to apply these methods using the machinery of computer science, that is, programming, one needs to unpack the process of these methods and schemes, look for efficient or sometimes simpler approaches, recognize patterns and develop strategies for suitable solutions. The more often one engages in such exercises, the more ingrained are these habits of mind.

## Examples

In this section, we discuss three examples of modelling tasks. In each case, the solution process is described, and the link to the "good habits" that one could possibly have developed from computer programming or computing exercises is made.

### *Example 1: Dealing with data (Tension-Strain relationship)*

In an experiment to study the elastic properties of arteries carried out under laboratory conditions, a human external iliac artery subject to some force (tension, $T$) and the response in terms of the stretch (strain, $a/a_0$, where $a$ is the extended length and $a_0$ is the original length of the artery) was measured [10]. The results are shown in the Table 1 below.

Table 1. Tension and strain in a human iliac artery

| $T$ (g/cm) | 4 | 8 | 13 | 20 | 22 | 28 | 33 | 40 | 44 | 60 | 71 | 83 | 95 | 109 | 132 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| $a/a_0$ | 1.22 | 1.35 | 1.45 | 1.50 | 1.55 | 1.57 | 1.60 | 1.64 | 1.67 | 1.71 | 1.74 | 1.77 | 1.80 | 1.83 | 1.86 |

Suppose it is required to obtain a function to represent the relationship between the two quantities (tension, or stress, and strain) using these experimental data points. The reason could be that a closed-form function is necessary in order to model blood flow through an artery, for instance. Now, someone with the habit of looking for patterns in problem solving, might want to first plot a graph of the points. The graph of $T$ against $a/a_0$ is as shown in Figure 1 below.

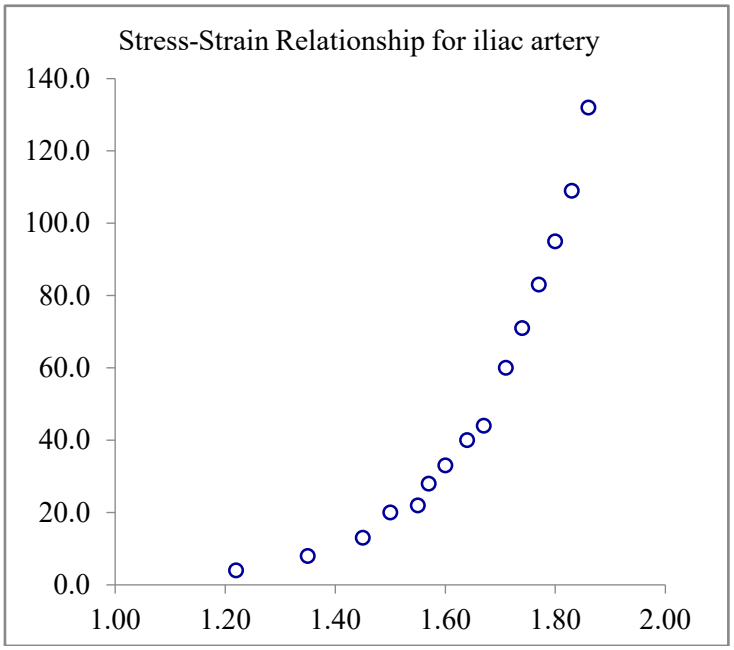



Figure 1. Plot of experimental data points showing stress-strain relationship of an iliac artery

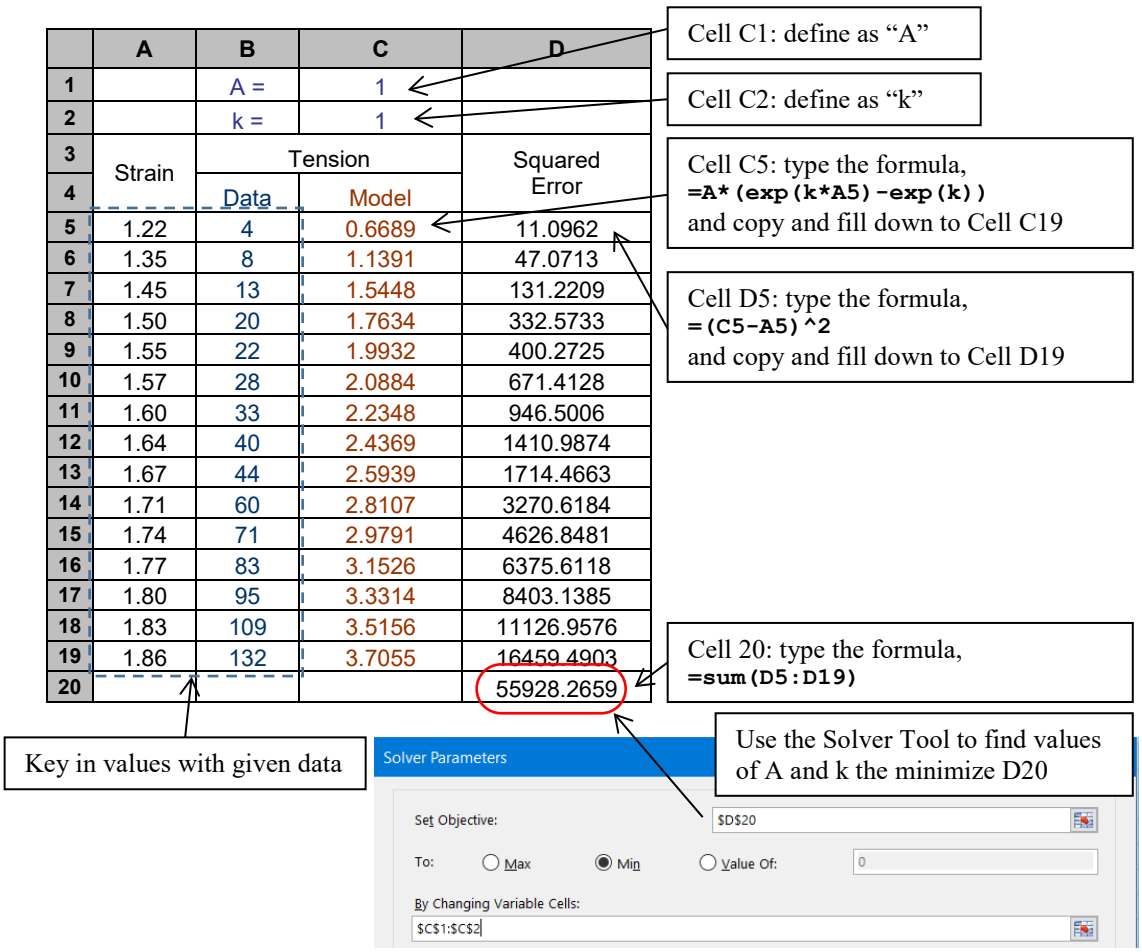
From the plot of the points, it would be natural to identify possible types or classes of functions that may be suitable to represent this set of points. In one such modelling case, the suggested function takes the form,

$$T = A(e^{kx} - e^{k})$$

where $A$ and $k$ are parameters to be found using the data set [7]. One way to find these parameters would be the least squares method, or some regression techniques, and would involve minimizing some form of errors in fitting the curve to the data set.

Because there are two parameters involved and they are related in a non-linear fashion, a more complicated technique involving iterating and improving subsequently found values of $A$ and $k$ has been suggested in the aforementioned paper. At each iteration, as described in the paper, the sum of residual squares (SRS), defined as the sum of the squares of the difference between the each data value and model value, is minimized for parameters $A$ and $k$ found at that stage.

The idea of minimizing errors in curve-fitting is so common that it has become a habit among researchers using numerical techniques. In this particular case, the problem of finding these two parameters that best fit the curve to the data can be solved using, for instance, the *Solver* tool in Microsoft Excel, described diagrammatically in Figure 2 below.




| | A | B | C | D |
|---|---|---|---|---|
| 1 | | A = | 1 | |
| 2 | | k = | 1 | |
| 3 | Strain | Tension | | Squared Error |
| 4 | | Data | Model | |
| 5 | 1.22 | 4 | 0.6689 | 11.0962 |
| 6 | 1.35 | 8 | 1.1391 | 47.0713 |
| 7 | 1.45 | 13 | 1.5448 | 131.2209 |
| 8 | 1.50 | 20 | 1.7634 | 332.5733 |
| 9 | 1.55 | 22 | 1.9932 | 400.2725 |
| 10 | 1.57 | 28 | 2.0884 | 671.4128 |
| 11 | 1.60 | 33 | 2.2348 | 946.5006 |
| 12 | 1.64 | 40 | 2.4369 | 1410.9874 |
| 13 | 1.67 | 44 | 2.5939 | 1714.4663 |
| 14 | 1.71 | 60 | 2.8107 | 3270.6184 |
| 15 | 1.74 | 71 | 2.9791 | 4626.8481 |
| 16 | 1.77 | 83 | 3.1526 | 6375.6118 |
| 17 | 1.80 | 95 | 3.3314 | 8403.1385 |
| 18 | 1.83 | 109 | 3.5156 | 11126.9576 |
| 19 | 1.86 | 132 | 3.7055 | 16459.4903 |
| 20 | | | | 55928.2659 |

Figure 2. Using the Solver Tool in Excel to find best parameter values

In this case, the Solver tool produces the values of $A \approx 0.0074$ and $k \approx 5.26$, which are also the values obtained in [7].

*Example 2: Simulating a process (Broken spaghetti problem).*

If a stick of spaghetti is slowly bent and broken at two points, the outcome is three shorter sticks. Suppose these two points are randomly chosen, what is the probability that the resulting three shorter parts will form a triangle?

Intuitively, we can figure out that not every three segments of the spaghetti will form a triangle. This is illustrated in the examples shown in Figure 3.

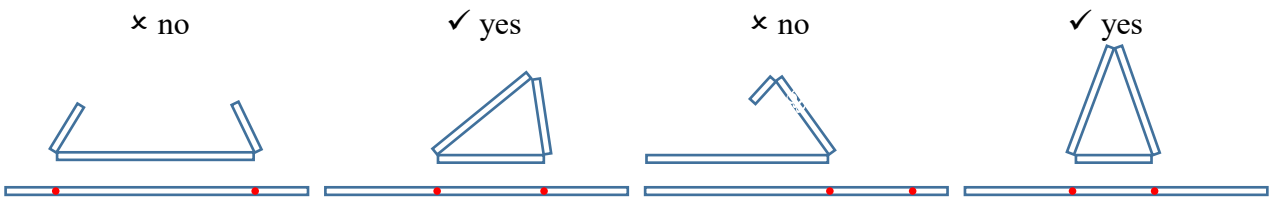



Figure 3. Examples of broken spaghetti segments forming and not forming a triangle

In general, suppose we have three segments with measuring $a$, $b$ and $c$ in length, and without loss of generality, assume $c$ is the longest segment. It is clear that in order for these three segments to form a triangle, then $a + b > c$. Further analysis along this line can be carried out to find the theoretical probability of forming a triangle from a broken spaghetti [1, pp 84–85].

On the other hand, it is also possible to approach this problem through a simulation model, which is in fact a form of a computational model. One could simulate the process of randomly choosing two points on a stick of spaghetti of unit length, say, and then record the lengths of the three resulting segments. If the length of the longest segment exceeds half the length of the original spaghetti stick, then a triangle cannot be formed. This is because the sum of the lengths of the other two shorter segments will be less than that of the longest segment and obviously a triangle cannot be formed. Otherwise, we increase the count of the number of triangles formed. This is done iteratively until we have sufficient number of runs to establish the experimental probability required. Such a simulation model could be implemented through the algorithm depicted as a flowchart in Figure 4. Using the flowchart, it would not be difficult to write the code, or use a spreadsheet like Microsoft Excel, to run the simulation to obtain the experimental probability of the three broken spaghetti segments forming a triangle.

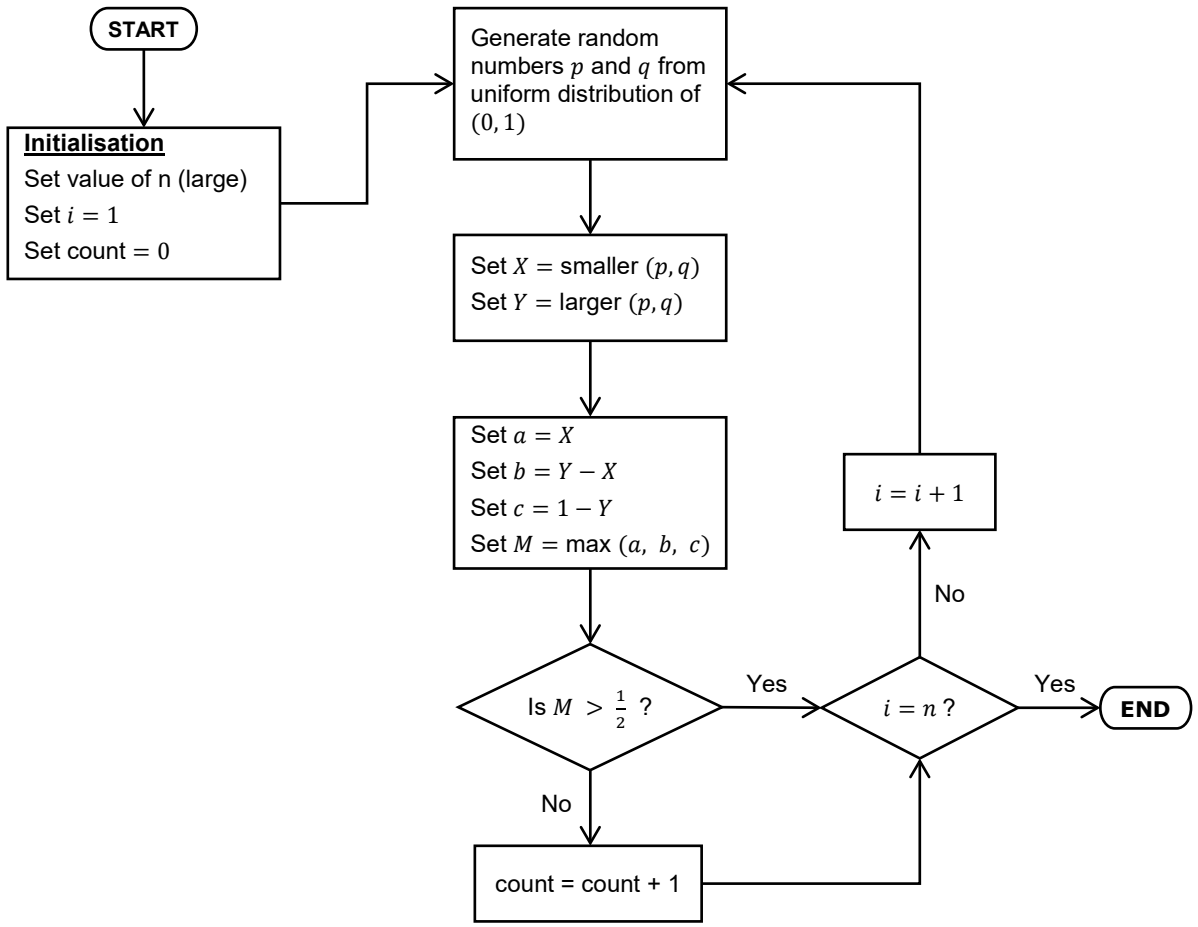



Figure 4. Flowchart showing algorithm to simulate the process in the broken spaghetti problem

Figure 5 shows how a simple Excel spreadsheet is set up, using form controls such as buttons and scroll bars to control events in the sheet. The scroll bar controls and sets the number of spaghetti sticks in this simulation, and this essentially means the number of trials used in one run of the simulation. On the right is the code written in Visual Basic for Applications (VBA) in Excel. The function "Sub Run()" is linked to the button labelled as "Run Simulations" on the Excel sheet. The program starts by assigning a variable ("n") to the value in Cell B2 and initializing the count value to zero. Random numbers are generated to simulate the breaking at two random points of a stick of unit length, and then assigning the values of lengths a, b and c of the resulting segments. The maximum is then found using a worksheet function, and used to determined if a triangle can be formed. Results of the experimental probability is copied to Cell B4.

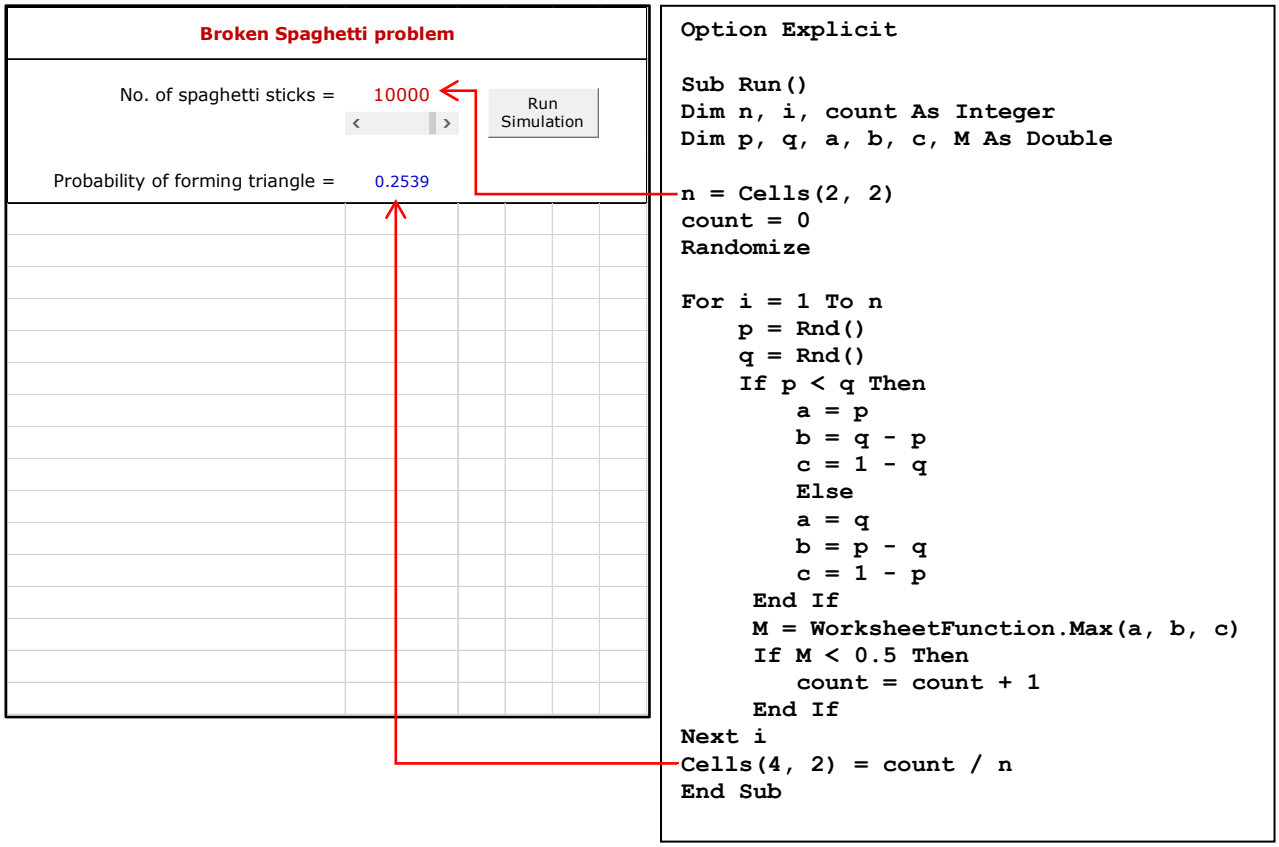



Figure 5. Set up of spreadsheet in Excel and VBA code to simulate the broken spaghetti problem

Two key features in this simulation model is the use of the law of large numbers and the concept of iteration. The law of large numbers basically says that if an experiment is performed a large number of times, the results obtained should be close to the expected value and the higher the number of trials, the closer will these results be to the expected value. This law is often used in simulations and very often, the "experiments" are computational experiments carried out via a computer program. In addition, in programming terms, to repeatedly carry out an experiment simply means to iterate a process a certain number of times. The idea of iteration, or repetition of events, therefore goes hand in hand with simulation models. The use of the law of large numbers and concept of iteration are often a second nature or an ingrained habit to a computer programmer tasked with implementing a simulation model.

*Example 3: Making a decision (Resource allocation problem)*

In many organizations, such as an academic department of a university, funds are allocated to their members on an annual basis for professional development. Very often, there are rules associated to the utilization of such funds. Suppose an academic department is allocated a fixed annual budget based on the number of faculty members for professional development purposes, given the rules and constraints, as well as the differing demands of the faculty members, how does one utilize the funds optimally while ensuring a fair distribution to members?

As an example, suppose at the start of the year, the department is given a certain budget, $P$ , to be used for professional development of its staff. Members of the department may submit requests for funding (to attend courses, conferences, workshops, and so on), typically of differing amounts, and we assume that there are $n$ members and each member may only make one request per year. If the total amount requested exceeds the $P$, then not every member will be able to obtain their requested funding amount. At the same time, optimal use of the budget would also require that the total utilization of the funds be as close to $P$ as possible. In other words, there should be as little unutilized funds as possible. Mathematically, we could represent the situation as follows.

Suppose member $i$ makes a request of $M_i$ amount of funds, and the approved amount is a actually $k_i M_i$ where $k_i \in (0, 1)$, then the objective would be that we need to minimize the quantity,

$$R = P - \sum_{i=1}^{n} k_i M_i \ ,$$

subject to the condition, $\sum_{i=1}^{n} k_i M_i \leq P$. This means that essentially, we need to find the set of values of $k_i \in (0, 1)$ so that $R$, which should be positive, is as small as possible.

The process or model may be simplified by setting a baseline for the approved amount. For instance, every member should be given at least a certain fixed fraction, $k$ of the requested amount. In addition, depending on other factors, a member could be given an integer multiple, called a weight, of a certain "step" $\Delta k$ of the requested amount. Hence, we have

$$k_i = k + \alpha_i \, \Delta k \ .$$

While $k$ may be set to ensure condition of not exceeding the given budget is met, each individual member $i$ will need to have its weight $\alpha_i$ determined in some manner. How these are assigned could depend on various factors, such as whether the member has been given larger amounts in previous years, or whether the member is a more junior staff member more deserving of professional development, and so on. Assuming that $k$ is suitably set, and $\alpha_i$ are appropriately assigned, then the problem reduces to finding the value of $\Delta k$ that minimizes $R$.

Consider the following hypothetical case with ten members as shown in Table 2. Each of the ten (fictitious) individuals' requested amount and corresponding weight assigned are given in columns 3 and 4. The total amount of requests is \$27,560. Note that the weights assigned here are hypothetical. In this case, for instance, it is assumed that two members, Ali and Goh, deserve a heavier weight (value of 5) for some reason, which could be related to their positions in the department, their previous funding requests, or other contributing factors. On the other hand, three other members deserve only the smallest weight value of 1, again for some supporting reasons.

Table 2. Amount of funding requested, and amount eventually allocated

| No. | Name | Requested Amount in \$ ($M_i$) | Weights ($w_i$) | $k_i$ | Allocated Amount in \$ ( $k_i \times M_i$ ) |
|---|---|---|---|---|---|
| 1 | Ali | 2,600 | 5 | 0.79 | 2,058.68 |
| 2 | Bai | 3,210 | 1 | 0.56 | 1,792.34 |
| 3 | Chandu | 4,160 | 2 | 0.62 | 2,565.55 |
| 4 | Darren | 1,890 | 1 | 0.56 | 1,055.30 |
| 5 | Esther | 3,610 | 1 | 0.56 | 2,015.68 |
| 6 | Faridah | 2,520 | 5 | 0.79 | 1,995.33 |
| 7 | Goh | 1,540 | 3 | 0.68 | 1,039.62 |
| 8 | Han | 1,980 | 2 | 0.62 | 1,221.10 |
| 9 | Ismail | 2,950 | 4 | 0.73 | 2,163.65 |
| 10 | Jan | 3,100 | 3 | 0.68 | 2,092.75 |
| | Total | 27,560 | | Total | 18,000.00 |

Suppose the budget is \$18,000 (that is, $P = 18000$), and assume that each member should at least be given 50% of what they have requested (that is, $k = 0.5$). Given this scenario and the set of values, we can again make use of Excel's Solver Tool and find the value of $\Delta k$ that minimizes $R$. It turns out that in this case, when $\Delta k = 0.0584$, the $R \approx 0$. With this value of $\Delta k$, all the $k_i$ may be computed, and the corresponding allocated amount are found. These are shown in the last two columns in the table.

The habits of mind invoked in this example are those concerned with decomposing a problem, breaking it down into smaller, more manageable parts, and then systematically building a model. For instance, given an open problem like in this example, as in programming, one first needs to identify the variables involved and determine the relationships among these variables. In this case, the overall budget, the individual members' requested amounts, the weights that need to be assigned, and the size of the additive "step". This example also illustrates how to simplify a problem, a process of reduction, and cut down the number of variables so that the problem can be more easily dealt with.

## Discussion

Any effort to develop thinking (such as mathematical thinking, creative thinking, critical thinking, and so on) should naturally be associated with some skills and competence, and constant use and practice of these skills over a sustained period of time. In the case of computational thinking, it has been suggested that these skills have to do with computer programming. Here, computer programming is more than just writing code – it refers to the process of designing and implementing solutions to problems that can be carried out on or by a computer.

Figure 6 shows a schematic of some of the key concepts and ideas in a typical introductory course in computer science or computer programming. These include learning about keeping to the rules and syntax of a programming language, process, flowcharts, control structures, recursion, data structure, variables, sorting algorithms, subroutines, and so on.

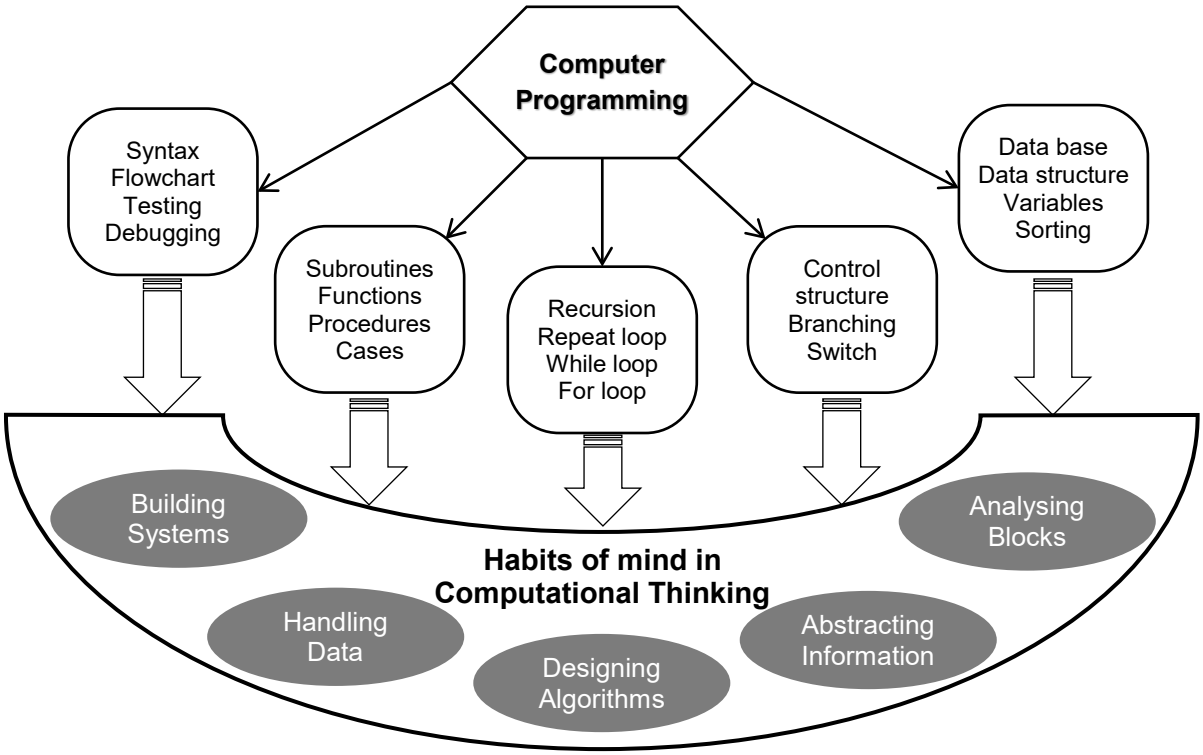



Figure 6. Concepts and skills in computer programming leading to the development of habits in computational thinking

It is also typical of such courses to require students to complete programming exercises to reinforce these concepts, achieve a deeper understanding, and be skillful in using them. Over time, as the student does more of these exercises and uses these concepts more frequently, it is not unreasonable to expect these skills to become internalized as habits that the student is likely to rely on in problem situation.

While it is difficult to say which skills or concepts develop what aspect of computational thinking, it is not hard to see that these can lead to developing some of the habits in Figure 6. For instance, constantly working with flowcharts, subroutines, functions and procedures in writing programs and solving problems serves to train one's mind to think systematically, visualize a big picture, while taking care to analyze smaller units and blocks in the solution process. At the same time, working with data and defining or creating variables and data structures are part and parcel of computer programming.

These skills eventually build one's familiarity with methods of looking at data and finding ways to use them effectively in problem solving, as is also demonstrated in Example 1 in the preceding section. Simulation models can be effectively designed and implemented if one has the knowledge and habit of thinking algorithmically, as illustrated in Example 2. Finally, abstracting information and describing them as variables in a computer program helps develop one's mind to think of factors in the real world as mathematical variables, as shown in Example 3. It is evident that these are the type of thinking and habits of mind that have eventually led to successful designs of solutions, implementable on a computing machine in the examples discussed.

## Concluding Remarks

In this paper, we examine and explicate aspects of habits of mind related to computational thinking, so that we may see their relevance in mathematical modelling. Using three examples, these habits or dispositions towards problem solving are teased out and linked to certain parts or phases in the solution process of the mathematical modelling tasks.

It is *not* the intent of this paper to advocate or even suggest that computational thinking is important for mathematics, nor is it its purpose to discuss related issues in mathematics education. It is acknowledged that in this respect, opinions do differ, and the diverse views on computational thinking and its relevance to mathematics education, coupled with the lack of research in this area, have meant that a common understanding does not seem to exist at the moment. Nonetheless, what this paper attempts to do is to illustrate and elucidate the link between some habits developed consciously through purposeful computer programming exercises and certain skill sets useful to mathematical modelling.

These mental habits of computer programmers or scientists, however, are not something that can be acquired through a few hours of coding lessons or even one course on a programming language. Rather, these are developed through many hours of computer programming exercises or projects, and frequent use of relevant programming skills and techniques in problem solving. In terms of mathematical modelling, these habits will further strengthen one's ability and competence in handling and solving modelling tasks.

## References

[1] Ang, K.C. (2018). *Mathematical Modelling for Teachers: Resources, Pedagogy and Practice*. London: Routledge.

[2] Ang, K.C. (2019). Exploring the nexus between computational thinking and mathematical modelling, paper presented at *The 19th International Conference on the Teaching of Mathematical Modelling and Applications*, Hong Kong.

[3] Costa, A. L. and Kallick, B. (2008). Learning and leading with habits of mind: 16 essential characteristics for success. Alexandria: Association for Supervision and Curriculum Development.

[4] Goldenberg, E. P. (1996). 'Habits of Mind' as an Organizer for the Curriculum. *Journal of Education*, 178 (1), 13–34.

[5] Hu, C. (2011). Computational thinking – what it might mean and what we might do about it, *Proceedings of the 16th annual joint conference on innovation and technology in computer science education*, 223–227. Darmstadt, Germany.

[6] Mannila, L., Dagiene, V., Demo, B., Grgurina, N., Mirolo, C., Rolandsson, L. and Settle, A. (2014). Computational thinking in K-9 Education, *Proceedings of the 19th annual joint conference on innovation and technology in computer science education*, 1–28. Uppsala, Sweden.

[7] Mazumdar, J., Ang, K. C. and Soh, L. L. (1991). A mathematical study of non-Newtonian blood flow through elastic arteries, *Australasian Physical & Engineering Sciences in Medicine*, 14(2), 65–73.

[8] Mohaghegh, M. and McCauley, M. (2016). Computational thinking: the skill set of the 21st century, *International Journal of Computer Science and Information Technologies*, 7(3), 1524–1530.

[9] Nardelli, E. (2019). Do we really need computational thinking? *Communications of the ACM,* 62(2), 32–35.

[10] Roach, M. R. and Burton, A. C. (1957). The reason for the shape of the distensibility curves of arteries, *Canadian Journal of Biochemistry and Physiology*, 35, 681–690.

[11] Sanford, J. (2013). Core concepts of computational thinking, *International Journal of Teaching and Case Studies*, 4(1), 1–12.

[12] Sanford, J. F. and Naidu, J. T. (2016). Computational thinking concepts for grade school, *Contemporary Issues in Educational Research*, 9(1), 23–31.

[13] Weintrop, D., Beheshti, E., Horn, M., Orton, K., Jona, K., Trouille, L. and Wilensky, U. (2016). Defining computational thinking for mathematics and science classrooms, *Journal of Science Education and Technology*, 25, 127–147.

[14] Wing, J. (2006). Computational Thinking, *Communications of the ACM,* 19(3), 33–35.

[15] Yasar, O. (2018). A new perspective on computational thinking, *Communications of the ACM,* 61(7), 33–39.