

Integrating Computational Thinking into Science Instruction: A Design-based Approach to Learning Protein Synthesis

Ladapa Ladachart¹  and Luecha Ladachart^{2*} 

¹Department of Curriculum, Teaching, and Learning, Faculty of Education, Chiang Mai University, Thailand

²Department of Curriculum and Instruction, School of Education, University of Phayao, Thailand

ABSTRACT

Introduction: Many countries consider the expansion of the workforce in science, technology, engineering, and mathematics (STEM) a key strategy for economic growth and social development. This belief applies in Thailand, with integration of STEM education at the K–12 level promoted in the most recent version of the national science curriculum standards. Engineering design and computational thinking have been introduced to the curriculum in addition to the canonical sciences (e.g., physics, chemistry, biology, astronomy, and geology). Although design-based learning has been proposed nationally as an integrated approach to STEM education, these new strands present challenges to science teachers. Many of these teachers struggle to envision how engineering design and computational thinking can be meaningfully integrated into science instruction, highlighting a clear need for concrete examples.

Content: This academic article presents an example of how engineering design and computational thinking can be incorporated into science instruction, illustrating an integrated approach to STEM education. Using the 6E Learning byDeSIGN™ model of design-based learning developed by the International Technology and Engineering Educators Association (Burke, 2014), the article describes a pedagogical activity on the topic of protein synthesis, which is often considered difficult for students to learn and for science teachers to teach. In this activity, designed for the high school level, students are challenged to design computer software that identifies a series of amino acids forming a protein based on a given strand of deoxyribonucleic acid (DNA). This activity involves six steps: Engage, Explore, Explain, Engineer, Enrich, and Evaluate. Specifically, students learn what proteins are, what they do in the body, and what they are made of. Once students understand that each protein consists of amino acids arranged in a unique sequence, they explore and explain how proteins are synthesized from DNA using a computer simulation. With this understanding of the process through which a protein is synthesized, students are introduced to the Python programming language and learn the necessary

ARTICLE INFO

Article history:

Received 2 July 2025

Revised 20 November 2025

Accepted 9 January 2026

Keywords:

Biology Teaching,
Computational Thinking,
Design-based Learning,
Integrated STEM Education,
Protein Synthesis

concepts of computational science (e.g., variables, loops, conditionals, and algorithms). Students are then tasked with applying these computational concepts to develop the computer software. Finally, they discuss and reflect on what they have learned. Through this activity, students learn the key process of protein synthesis (e.g., transcription and translation) and discover how computational concepts are used to model biological processes effectively.

Conclusion: The 6E Learning byDeSIGN™ model of design-based learning is similar to the 5E model of inquiry-based learning—Engage, Explore, Explain, Elaborate, and Evaluate—which has long been promoted in Thailand and elsewhere. Science teachers are already familiar with the 5E inquiry-based model, so they can easily understand and adopt the 6E Learning byDeSIGN™ model. As such, the activity presented in this article serves as a practical example for science teachers to develop lessons that integrate engineering design and computational thinking into the teaching of other science topics. Yet, further research is needed to investigate its potential impact on students’ learning and science teachers’ professional development.

**Corresponding author*

E-mail address: ladachart@gmail.com

Introduction

Expanding the workforce in science, technology, engineering, and mathematics (STEM) has become a priority for many countries seeking to enhance economic growth and social development (Bacovic et al., 2022). Thailand shares this objective, as the development of human resources in STEM is included in its national economic and social development plan (Office of the National Economic and Social Development Council, 2022). According to this plan, Thailand envisions that increasing the STEM-related workforce will help the country transition from an agriculture- and industry-based economy (i.e., Thailand 1.0–3.0) to an innovation-driven one (i.e., Thailand 4.0), since STEM fields are fundamental to producing innovations that add value in the global market. This transition is expected to help Thailand overcome the middle-income trap that has hindered its economic progress for decades (United Nations, 2015). In this regard, STEM education at the K–12 level is considered a mechanism geared toward preparing new generations with the knowledge, skills, and attitude necessary to pursue STEM-related higher education and careers (Promboon et al., 2018).

In response to the mission outlined in the national economic and social development plan (Office of the National Economic and Social Development Council, 2022), Thailand’s national science curriculum standards have been revised to incorporate engineering design and computational thinking—alongside traditional sciences such as physics, chemistry, biology, astronomy, and geology—to promote integrated approaches to STEM education (Bureau of Academic Affairs and Educational Standards, 2017). These shifts toward an interdisciplinary

curriculum are intended to promote not only students' science learning, but also their ability to solve complex problems in the future. As evidenced by the recent COVID-19 pandemic, addressing emerging global challenges requires not only scientific knowledge and understanding, but also the application of engineering design and computational thinking (e.g., in developing effective vaccines). Therefore, in addition to increasing workers in STEM-related fields, STEM education at the K–12 level helps prepare new generations to engage with and adapt to unexpected challenges (Promboon et al., 2018).

Design-based learning has been proposed as an integrated approach to STEM education to support science teachers in implementing the curricular changes (Institute for the Promotion of Teaching Science and Technology, 2014). Despite its various forms (Ladachart & Ladachart, 2020), design-based learning typically involves stages such as “defining the problem and identifying the need, collecting information, introducing alternative solutions, choosing the optimal solution, designing and creating a prototype, and evaluation” (Doppelt et al., 2008, p. 24). Specifically, design-based learning follows an engineering design process that not only connects science, technology, and mathematics but also relates to students' everyday experiences (Quinn et al., 2020). Regardless of its specific form, design-based learning can facilitate students' STEM learning in various domains, including knowledge, skills, and attitude (Delen & Sen, 2023). However, given the epistemological and methodological differences between science and engineering, Pleasants et al. (2021) suggest that design-based learning must be carefully designed and structured to effectively facilitate students' science learning.

While engineering design is clearly emphasized in design-based learning (Ladachart & Ladachart, 2020), computational thinking—another new strand in Thailand's national science curriculum standards (Bureau of Academic Affairs and Educational Standards, 2017)—is not explicitly included. According to Wing (2006), “computational thinking involves solving problems, designing systems, and understanding human behavior, by drawing on the concepts fundamental to computer science”, and it is now considered “a fundamental skill for everyone, not just for computer scientists” (p. 33). Since computational thinking involves the mental activity of formulating a problem to admit a computational solution, it can be integrated into the design process and STEM education in order to meaningfully support students' STEM learning (Hsiao et al., 2022; Li et al., 2020). However, given the already crowded curriculum, researchers recommend using computational thinking “as a medium for teaching other subjects” (Grover & Pea, 2013, p. 42), such as science. Integrating computational thinking into design-based science learning thus represents a promising approach.

Nonetheless, the recommendation to integrate computational thinking into science instruction presents a key challenge for science teachers. For example, through their survey of science teachers' conceptualizations of computational thinking and its integration into science lessons, Kite and Park (2023) found that many lacked understanding of computational concepts and believed their students were not sufficiently prepared to learn science alongside computational thinking. This suggests that science teachers need professional development to support this integration (Yadav et al., 2018). As part of such professional development, Ogegbo and Ramnarain (2022) emphasize that science teachers require practical and contextual ideas on strategically embedding computational thinking in science instruction. Because this integration poses practical difficulties for science teachers who lack concrete examples—a challenge that seems even more pronounced when science is taught through design-based

learning—we aim to support science teachers in this endeavor through illustrating such a concrete example in this article.

Based on our personal experience in teaching biology and working with preservice teachers majoring in biology, we chose a biological topic. In biology, computational science has been applied in numerous ways; as Markowetz (2017) notes, “today all biology is computational biology” (p. 2). One of the well-known applications of computational thinking in biology occurs when dealing with big data, including those on genomes (Schatz, 2012). Given that the recent reform in science education emphasizes that science teachers teach science in ways that correspond to and reflect how scientists work (National Research Council, 2007), it would be suitable to teach students computational thinking to help them work with bioinformatics (Qin, 2009). While Peel and Friedrichsen (2018) demonstrate that protein synthesis affords a disciplinary context for integrating computational thinking into science teaching, their demonstration focuses on transcription rather than the whole process of protein synthesis. Extending their research scope, this article presents how the topic of protein synthesis can be taught alongside computational thinking through design-based learning.

To achieve our goal, the article is divided into two sections. In the first section, we present the fundamental elements relevant to the instructional activities. These include the main concepts of protein synthesis (e.g., transcription and translation), the key concepts of computational thinking (e.g., loops and conditionals), and the model of design-based learning used to develop the instructional activities (i.e., 6E Learning byDeSIGN™; Burke, 2014). In the second section, we describe the instructional activities in detail, including the writing of computer code in Python using JupyterLab to model the biological processes of protein synthesis. In doing so, we acknowledge that we made our decisions regarding the computational tools considering accessibility and practicality, while realizing that other tools might be more effective. We also leave decisions about the duration of each stage of the instructional activities to readers, as this depends on students’ prior knowledge and ability. We hope that the instructional activities provide science teachers with a practical guideline on how computational thinking can be taught and used as a medium to teach a scientific concept.

Fundamental Elements

Protein Synthesis

According to Thailand’s national science curriculum standards, protein synthesis is a biological concept that students are expected to learn in the tenth grade. Specifically, they must be able to “explain the relationship between genes, protein synthesis, and genetic traits” (Bureau of Academic Affairs and Educational Standards, 2017, p. 36). Thus, the instructional activities presented in this article were designed with students at this grade in mind. As an abstract concept involving substantial terminology, protein synthesis is considered difficult for students to comprehend (Guzman & Bartlett, 2012). Consequently, students often have several misconceptions of protein synthesis; for example, they may be confused about the specificity of related terms (e.g., protein, enzyme, amino acid), believe that amino acids are produced by translation, and/or misunderstand the dual roles of enzymes, which can be both the players and products of protein synthesis (Fisher, 1985). As a result, protein synthesis is considered a biological concept that is challenging for science teachers to teach (Koba & Tweed, 2009), who may themselves also hold such misconceptions similar to students.

Based on a review of relevant literature, we identified a number of instructional strategies that have been developed to facilitate students' learning of protein synthesis. These include model building, role playing, and concept mapping. Regardless of the strategy used, Koba and Tweed (2009) suggest that protein synthesis should be taught through a four-step sequence designed to gradually deepen students' understanding, as shown in Table 1. This suggestion aligns with the notion of teaching a scientific concept according to learning progression, which is defined as "descriptions of the successively more sophisticated ways of thinking about a topic that can follow one another as children learn about and investigate a topic over a broad span of time" (National Research Council 2007, p. 214). In other words, a complex concept in science (e.g., protein synthesis) should be taught in a progressively sophisticated way, evolving from a basic understanding toward more advanced knowledge acquisition. Following this notion, the sequence provided by Koba and Tweed (2009) forms the framework for the design the instructional activities presented in this article.

Table 1

Sequence for Teaching Protein Synthesis (Koba & Tweed, 2009)

Target	Understanding
1	Proteins execute the primary functions inside cells. They determine the structures and functions of organisms. The observable traits of an organism (phenotype) arise from the roles these proteins play.
2	Genes stored in DNA supply the genetic information necessary to create proteins. The sequence of nucleotides as subunits in DNA specifies how amino acids are arranged to form proteins.
3	The order of amino acids dictates the type and shape of the protein—and thus its function.
4	Mutations—changes in the DNA—impact protein production. Errors in the DNA can result in missing or inadequately functioning proteins, which results in a change in phenotype/trait.

Computational Thinking

Protein synthesis is a process that can be modeled using computational thinking (Peel & Friedrichsen, 2018). While computational thinking can be conceptualized in various ways, Shute et al. (2017) summarize it in terms of six facets: abstraction (i.e., extracting the essence of a complex problem), decomposition (i.e., dissecting a complex problem into manageable parts), algorithm (i.e., designing a procedure for solving a problem), debugging (i.e., detecting and fixing errors when a solution does not work as expected), iteration (i.e., repeating a process until a condition is met), and generalization (i.e., transferring computational ability to a wider range of situations). In this paper's context, protein synthesis as a biological process must be abstracted and decomposed into parts (i.e., transcription and translation), each of which can be further divided into smaller parts. Translation, for example, can be broken down to finding the start codon and repeatedly reading each codon—steps that lend themselves to designing algorithms that incorporate iteration. This process also involves debugging. Once developed, the algorithm for finding the start codon can be generalized to identify a stop codon.

Specifically, computational thinking entails several concepts, such as sequences, loops, parallelism, events, conditionals, operators, and data (Brennan & Resnick, 2012), as detailed in

Table 2. Students can use these concepts to model the protein synthesis, such as asking the user to enter DNA (i.e., events) and store it as a string in a variable (i.e., data). Students can also write code to repeatedly check whether each nucleotide in the DNA entered by the user consists of A, T, C, or G (i.e., loops), in order to process it; otherwise, the program informs the user that the DNA is incorrect (i.e., conditionals). To facilitate transcription and translation, students can also write code using a boolean expression to check for start and stop codons in the DNA (i.e., operators). Because these computational concepts are taught in elementary or lower secondary grades, students are likely to “apply [them] in developing a project that integrates with other subjects creatively and connects with real life” (Bureau of Academic Affairs and Educational Standards, 2017, p. 125). Based on this analysis, protein synthesis provides a potential context for students to apply computational thinking.

Table 2

Concepts in Computational Thinking (Brennan & Resnick, 2012)

Concept	Detail
Sequences	Ordered steps that the computer follows to carry out actions
Loops	Repetition of the same instructions for a specified number of times
Parallelism	Execution of multiple instructions at the same time
Events	Triggers that initiate actions and enable interactive behavior
Conditionals	Rules that allow different outcomes depending on conditions
Operators	Tools for performing mathematical or text-based operations
Data	Information that can be stored, accessed, and modified

Design-based Learning

Based on a constructionist perspective of learning, design-based learning has been used as an instructional model to integrate engineering design into science instruction (Delen & Sen, 2023; Doppelt et al., 2008). More recently, it has also been employed to support students’ development of computational thinking (Hsiao et al., 2022). Although design-based learning can take various forms (Ladachart & Ladachart, 2020), this article adopts the 6E Learning byDeSIGN™ model developed by the International Technology and Engineering Educators Association (Burke, 2014) to integrate computational thinking into science instruction on protein synthesis. As the name suggests, the 6E model has six stages: Engage, Explore, Explain, Engineer, Enrich, and Evaluate (see Table 3). Because its stages closely resemble those of the 5E inquiry cycle—an instructional model that has been nationally promoted and widely used in Thailand—science teachers are more likely to understand and adopt the 6E model than other models of design-based learning. Hence, this model was selected as an example to demonstrate the development of science lessons incorporating computational thinking.

Table 3*Six Stages of Design-based Learning (Burke, 2014)*

Stage	Purpose
Engage	To spark students' interest and activate their prior knowledge
Explore	To give students opportunities to investigate and construct understanding
Explain	To allow students to articulate and refine their developing ideas
Engineer	To deepen students' understanding by applying it in solving a problem
Enrich	To let students extend their learning to more complex situations
Evaluate	To assess the extent of student learning for both students and teachers

The six stages of the 6E model align well with Koba and Tweed's (2009) sequence for teaching protein synthesis. Specifically, with the aim of piquing students' interest, Engage focuses on what students know about proteins in the body and the functions of these proteins. Then, with the aim of helping students construct and explain their understanding of how each protein is synthesized to function in a specific way, Explore and Explain move them toward comprehending what proteins are made of, where they get the necessary information, and how the process unfolds. Subsequently, with the aim of enabling students to deepen their understanding through application, Engineer challenges them to design a computer program to model the biological process. Afterward, with the aim of enhancing students' understanding, Enrich draws attention to errors that can occur in both the computer program and the body. Finally, with the aim of assessing students' understanding, Evaluate encourages them to examine the similarities and differences between the computer program and the biological process. The next section illustrates how these pedagogical ideas can be realized in practice.

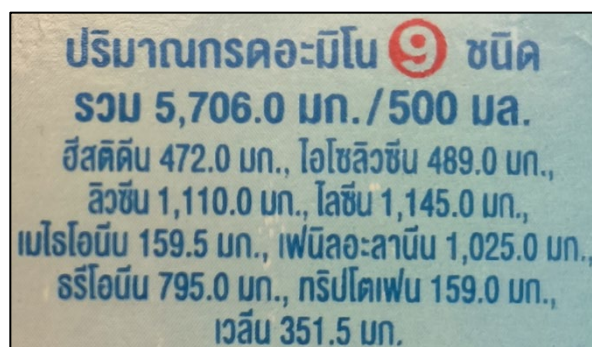
Instructional Activities

Engage

According to Koba and Tweed's (2009) framework for teaching protein synthesis, students should first understand that their bodies contain many types of proteins, each with distinct functions. To engage students, the instructional activities commence with asking whether they know what protein is and where it can be found. Drawing on their prior knowledge from elementary education, students are likely to focus on proteins in food (e.g., meat) rather than in the body. After shifting their focus to proteins within the body, students may recall that protein helps repair damage (e.g., wounds). Building on this idea, students can then be asked to search the internet for other protein functions in the body. This activity may yield results similar to those shown in Table 4. Once students have a basic understanding of the types and functions of proteins, they can be asked what components these various proteins have in common. To facilitate students' responses, they may be encouraged to examine a milk carton label (e.g., Figure 1), through which they can notice that protein consists of amino acids. Students can also share the names of amino acids to highlight their variety.

Table 4*Types and Functions of Proteins*

Type	Function	Example
Structural	To provide physical support and structure	Keratin provides strength in nails.
Contractile	To enable muscle movement	Tropomyosin regulates muscle contraction.
Transport	To move vital substances through the body	Albumin transports fatty acids in blood.
Storage	To maintain nutrients for later use	Ferritin stores iron in the liver and spleen.
Hormone	To regulate metabolic and nervous-system processes	Insulin controls blood glucose levels.
Enzyme	To speed up chemical reactions in cells	Lactase digests lactose in the small intestine.
Protection	To identify and remove harmful substances	Immunoglobulin protects the body from infections.

Figure 1*A Milk Carton Label Showing Some Amino Acids*


ปริมาณกรดอะมิโน	ชนิด
รวม 5,706.0 มก./500 มล.	
ฮีสติดีน 472.0 มก.,	ไอโซลิวซีน 489.0 มก.,
ลิวซีน 1,110.0 มก.,	ไลซีน 1,145.0 มก.,
เมไทโอนีน 159.5 มก.,	เฟนิลอะลานีน 1,025.0 มก.,
อาร์จินีน 795.0 มก.,	ทรีปโตเฟน 159.0 มก.,
เวอรีน 351.5 มก.	

After identifying some amino acids from the label, students can be asked to search the internet for additional types. They may discover that there are 20 amino acids, which can be categorized as essential and non-essential. The former are what the human body cannot construct but must obtain from food. The latter can be constructed by the body. At this point, students should be informed that all proteins, regardless of their type and function, are composed of a sequence of amino acids, each unique in number and order. While some protein (i.e., glutathione) contains only three amino acids, others (e.g., titin) can have more than 30,000. Based on this information, students can be asked how the body “knows” the correct number and order of amino acids when synthesizing a protein. This question is intended to direct students’ attention to DNA as the genetic code within the body. As students have learned about cell structures—including nucleus, chromosomes, DNA, and genes that determine genetic traits—in their lower secondary grades, they are likely to view DNA as the source that determines the number and sequence of amino acids to constitute a protein.

Explore

Once students have identified DNA as the source the body uses to determine the number and order of amino acids in protein synthesis, they may be asked about its structure and components. Students may explore a model of DNA, whether physical or graphical, to observe that it is a double-helix composed of four types of nitrogenous bases: adenine (A), thymine (T), cytosine (C), and guanine (G). They should be informed that a typical human cell contains about 6.4 billion nitrogenous bases and that each individual has a unique sequence of these bases. Based on this information, students can then be asked whether the order of amino acids in a protein relates to the order of nitrogenous bases in DNA. To answer this question, they can be encouraged to watch a video clip, such as one from the YourGenome channel on YouTube, that illustrates the process of protein synthesis. While watching, they should record which organelles are involved and what happens at each step of the process for classroom discussion. A computer simulation, such as one from the PhET Interactive Simulations project at the University of Colorado, can also be used to allow students to explore and test their ideas.

Explain

After watching the video clip and exploring the computer simulation—possibly several times—each student should be asked to describe the key steps of protein synthesis are. At this point, they should understand that it involves transcription and translation. Transcription occurs in the nucleus, where DNA is located. The sequence of DNA representing a gene is copied into messenger ribonucleic acid (mRNA), based on the pairing of two nitrogenous bases (e.g., C pairs with G, while A pairs with T). Unlike DNA, where thymine (T) pairs with adenine (A), mRNA uses uracil (U) for this pairing. Consequently, mRNA's composition includes the four nitrogenous bases: A, U, C, and G. Once the mRNA is constructed, it leaves the nucleus and enters the cytoplasm for translation. Translation occurs when a ribosome attaches to the mRNA and reads three nitrogenous bases—codons—simultaneously, with AUG as the start codon. Transfer ribonucleic acid (tRNA) then transports amino acids to the ribosome according to the codon table (see Figure 2). The ribosome links the amino acids together using peptide bonds in the correct order to create a protein until it finds one of the stop codons.

Figure 2

Codon Table

		Second letter				
		U	C	A	G	
First letter	U	UUU } Phe UUC } UUA } Leu UUG }	UCU } UCC } Ser UCA } UCG }	UAU } Tyr UAC } UAA Stop UAG Stop	UGU } Cys UGC } UGA Stop UGG Trp	U C A G
	C	CUU } CUC } Leu CUA } CUG }	CCU } CCC } Pro CCA } CCG }	CAU } His CAC } CAA } Gln CAG }	CGU } CGC } Arg CGA } CGG }	U C A G
	A	AUU } AUC } Ile AUA } AUG Met	ACU } ACC } Thr ACA } ACG }	AAU } Asn AAC } AAA } Lys AAG }	AGU } Ser AGC } AGA } Arg AGG }	U C A G
	G	GUU } GUC } Val GUA } GUG }	GCU } GCC } Ala GCA } GCG }	GAU } Asp GAC } GAA } Glu GAG }	GGU } GGC } Gly GGA } GGG }	U C A G

Source: Open Oregon (2019)

Once students can sufficiently explain the process of protein synthesis in their own words, they can discuss and identify which steps in this biological process can be modeled using concepts in computational science (e.g., sequences, loops, conditionals, and operators). Through this discussion, they can realize the correspondence between the two and begin mapping each biological step to a computational concept. For example, during transcription, students can apply conditionals (i.e., if-else statements) to represent the pairing of nitrogenous bases—such as replacing adenine (A) in DNA with uracil (U) in mRNA. Loops (e.g., for or while loops) can be used to repeat this base-pairing process until the end of the DNA strand is reached. Similarly, students can use conditionals to determine whether translation will begin—for instance, starting translation only if the start codon (i.e., AUG) is present in mRNA. Loops can be used to represent the process of adding amino acids until a stop codon is encountered. Figures 3 and 4 illustrate examples of flowcharts for transcription and translation, respectively. Joint creation of these flowcharts will consolidate students' understanding.

Figure 3

Flowchart of Transcription

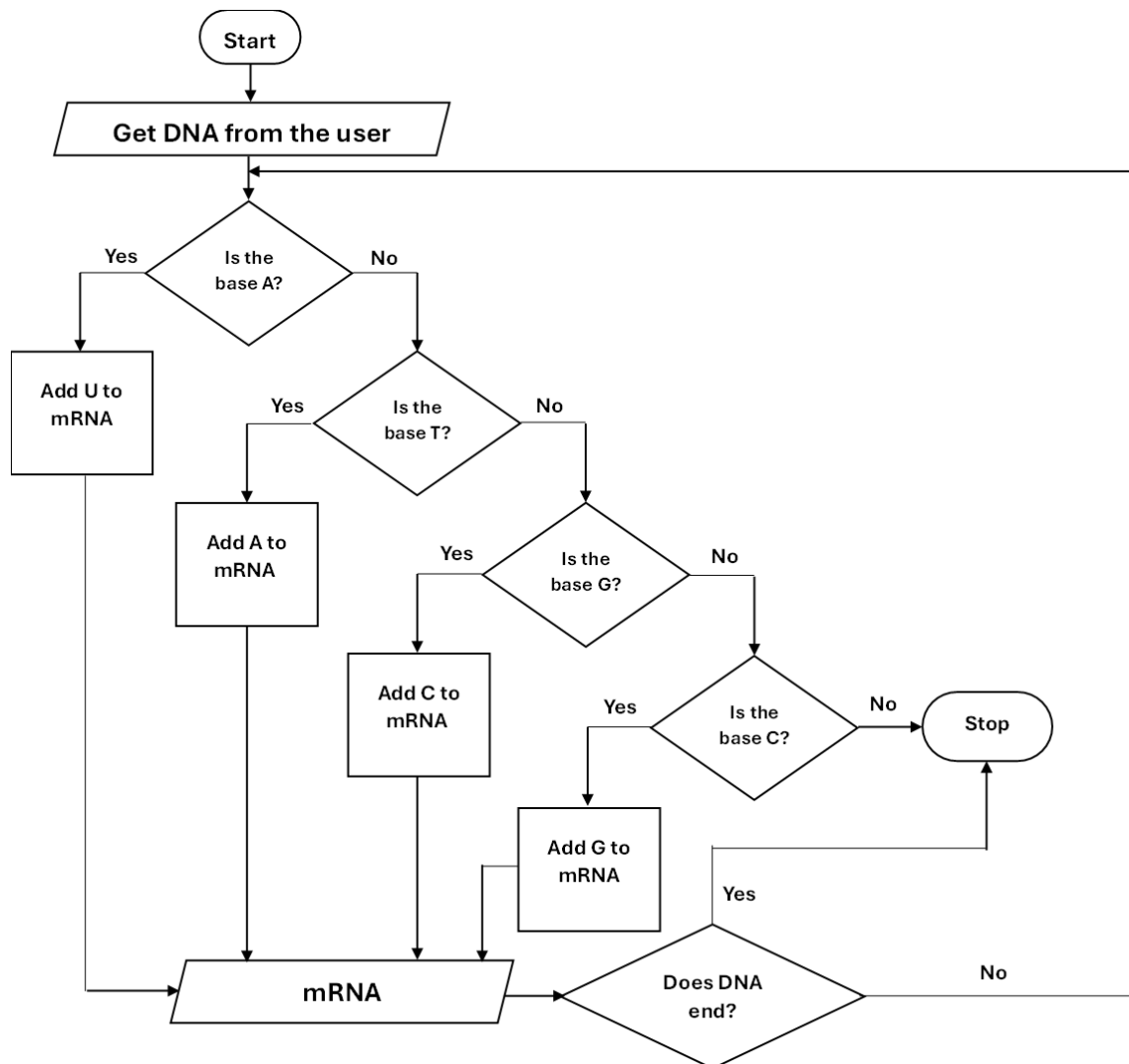
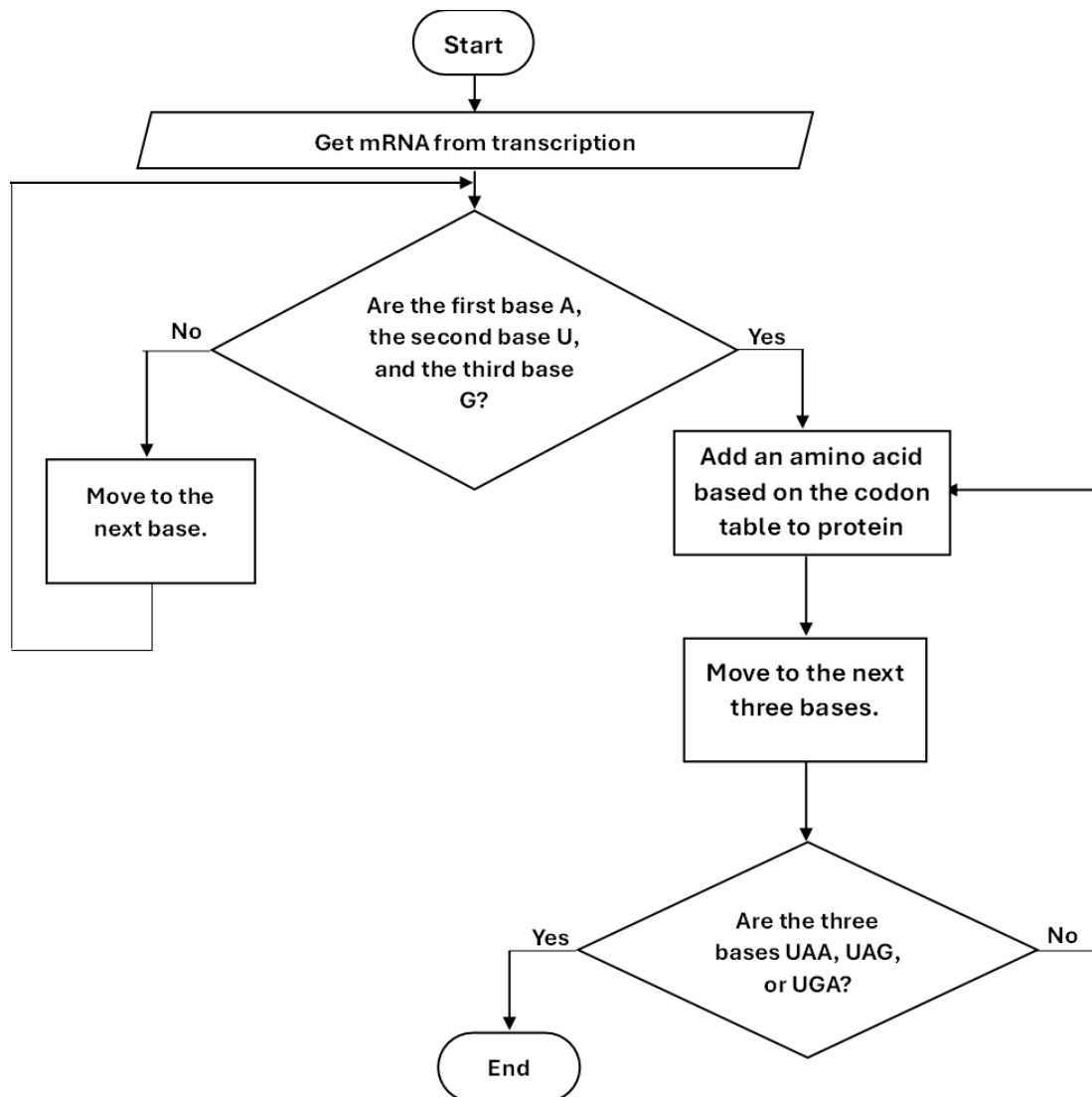


Figure 4*Flowchart of Translation***Engineer**

Using flowcharts that reflect their understanding of protein synthesis, students can design a computer program that identifies the order of amino acids from a given strand of DNA. While various programming languages and platforms are suitable for this purpose, this article uses Python as the programming language and uses JupyterLab within the Anaconda distribution as the programming platform, both of which are freely available. Using this platform, students can write their code directly in a web browser on their computer. To support this activity, students should be pointed to a website, such as RANDOM DNA SEQUENCE GENERATOR developed by the University of California, Riverside, through which they can generate a random strand of DNA. They can then use this DNA to explore various computational concepts. These include creating an event to get a DNA input from the user, defining a variable to store the DNA as a string, displaying the stored DNA, manipulating the DNA sequence, identifying its length, accessing specific characters, creating conditional statements, using loops, and applying logical expressions to produce true-or-false outcomes.

For example, students can write the code “DNA = str(input("Please enter DNA:"))” to prompt the user to enter a DNA sequence. Because Python is case-sensitive, they can also write the code “DNA = DNA.upper()” to ensure all nitrogenous bases in the DNA are in uppercase for further processing. To verify that the DNA sequence contains only valid nitrogenous bases (i.e., A, T, C, and G), students can write the code using a “for” loop to check whether each nitrogenous base is one of these four. If all nitrogenous bases are valid, the program will confirm the user and display the DNA sequence; otherwise, it will inform the user that the DNA is not valid and not proceed with further steps. Figure 5 shows the results for the cases of different input. Once this validation is successful, students can proceed to write the code for transcription. To do so, students can similarly apply a “for” loop to replace A with U, T with A, C with G, and G with C, as illustrated in Figure 6. After creating the mRNA, students can write the code to check whether and where the start codon is present. As shown in Figure 7, they can also apply another “for” loop to examine the mRNA base by base.

Figure 5

Code That Prompts the User for Input in Gray with Output in White

```
DNA = str(input("Please enter DNA:"))
DNA = DNA.upper()
for i in range(len(DNA)):
    if DNA[i] == "A" or DNA[i] == "T" or DNA[i] == "C" or DNA[i] == "G":
        valid_DNA = True
    else:
        valid_DNA = False
if valid_DNA == True:
    print("DNA is", DNA)
else:
    print("DNA is not valid.")
```

Please enter DNA: TACCCATGCC
DNA is TACCCATGCC

```
DNA = str(input("Please enter DNA:"))
DNA = DNA.upper()
for i in range(len(DNA)):
    if DNA[i] == "A" or DNA[i] == "T" or DNA[i] == "C" or DNA[i] == "G":
        valid_DNA = True
    else:
        valid_DNA = False
if valid_DNA == True:
    print("DNA is", DNA)
else:
    print("DNA is not valid.")
```

Please enter DNA: TACCCATGCCaaaa
DNA is TACCCATGCCAAAA

```
DNA = str(input("Please enter DNA:"))
DNA = DNA.upper()
for i in range(len(DNA)):
    if DNA[i] == "A" or DNA[i] == "T" or DNA[i] == "C" or DNA[i] == "G":
        valid_DNA = True
    else:
        valid_DNA = False
if valid_DNA == True:
    print("DNA is", DNA)
else:
    print("DNA is not valid.")
```

Please enter DNA: TACCCATGCCzzz
DNA is not valid.

Figure 6*Code for Transcribing DNA to mRNA in Gray with Output in White*

```

mRNA = ""
for j in DNA:
    if j == "A":
        j = "U"
    elif j == "T":
        j = "A"
    elif j == "C":
        j = "G"
    elif j == "G":
        j = "C"
    mRNA = mRNA + j
print("mRNA is", mRNA)

Please enter DNA: ATAAACTGC
DNA is ATAAACTGC
mRNA is UAUUUUGACG

```

Figure 7*Code for Finding a Start Codon in mRNA in Gray with Output in White*

```

startCODON = False
for k in range(len(mRNA)):
    base1=mRNA[k:k + 1]
    base2=mRNA[k + 1:k + 2]
    base3=mRNA[k + 2:k + 3]
    if base1 == "A" and base2 == "U" and base3 == "G":
        startCODON = True
        break
    else:
        k = k+1
if startCODON == True:
    print("A start codon is found at", k + 1, "-", k + 3)
else:
    print("A start codon is not found.")

Please enter DNA: ATAATACTGC
DNA is ATAATACTGC
mRNA is UAUUAUGACG
A start codon is found at 5 - 7

```

```

startCODON = False
for k in range(len(mRNA)):
    base1=mRNA[k:k + 1]
    base2=mRNA[k + 1:k + 2]
    base3=mRNA[k + 2:k + 3]
    if base1 == "A" and base2 == "U" and base3 == "G":
        startCODON = True
        break
    else:
        k = k+1
if startCODON == True:
    print("A start codon is found at", k + 1, "-", k + 3)
else:
    print("A start codon is not found.")

Please enter DNA: ATAAACTGC
DNA is ATAAACTGC
mRNA is UAUUUUGACG
A start codon is not found.

```

Subsequently, students can write the code for translation. Since the nitrogenous bases before the start codon are not used in this process, students can remove them. To do this, they can write the code “gene = mRNA[k:]”, which specifies that only the nitrogenous bases starting from the start codon are used as the gene for translation. Because the location of a stop codon within the gene is unknown, students must use a “while” loop instead of a “for” loop, to identify the amino acids corresponding to each codon. According to the codon table (see Figure 2), the

start codon represents the amino acid “Met”, and students can define this amino acid at the beginning of the polypeptide. The “while” loop continues until the end of the gene is reached or a stop codon is met. During this iteration, amino acids are linked to form a polypeptide, which becomes a protein. If no stop codon is found in the gene, the program will inform the user that the translation does not occur due to the absence of a stop codon. Otherwise, the program will display the sequence of amino acids that form the protein. Figure 8 shows the code for translation. This stage represents the process as it usually occurs.

Figure 8

Code for Translating mRNA to Protein in Gray with Output in White

```

gene = mRNA[k:]
m = 0
n = 3
poly_pep = "Met"
valid_gene = False
while n < len(gene) + 3:
    codon = gene[m + 3: n + 3]
    if codon == "UAA" or codon == "UAG" or codon == "UGA":
        valid_gene = True
        print("A stop codon is found at", m + 4, "-", n + 4)
        break
    elif codon == "UUU" or codon == "UUC":
        amino = "Phe"
    elif codon == "UUA" or codon == "UUG" or codon == "CUU" or codon == "CUC" or codon == "CUA" or codon == "CUG":
        amino = "Leu"
    elif codon == "AUU" or codon == "AUC" or codon == "AUA":
        amino = "Ile"
    elif codon == "AUG":
        amino = "Met"
    elif codon == "GUU" or codon == "GUC" or codon == "GUA" or codon == "GUG":
        amino = "Val"
    elif codon == "UCU" or codon == "UCC" or codon == "UCA" or codon == "UCG" or codon == "AGU" or codon == "AGC":
        amino = "Ser"
    elif codon == "CCU" or codon == "CCC" or codon == "CCA" or codon == "CCG":
        amino = "Pro"
    elif codon == "ACU" or codon == "ACC" or codon == "ACA" or codon == "ACG":
        amino = "Thr"
    elif codon == "GCU" or codon == "GCC" or codon == "GCA" or codon == "GCG":
        amino = "Ala"
    elif codon == "UAU" or codon == "UAC":
        amino = "Tyr"
    elif codon == "CAU" or codon == "CAC":
        amino = "His"
    elif codon == "CAA" or codon == "CAG":
        amino = "Gln"
    elif codon == "AAU" or codon == "AAC":
        amino = "Asn"
    elif codon == "AAA" or codon == "AAG":
        amino = "Lys"
    elif codon == "GAU" or codon == "GAC":
        amino = "Asp"
    elif codon == "GAA" or codon == "GAG":
        amino = "Glu"
    elif codon == "UGU" or codon == "UGC":
        amino = "Cys"
    elif codon == "UGG":
        amino = "Trp"
    elif codon == "CGU" or codon == "CGC" or codon == "CGA" or codon == "CGG" or codon == "AGA" or codon == "AGG":
        amino = "Arg"
    elif codon == "GGU" or codon == "GGC" or codon == "GGA" or codon == "GGC":
        amino = "Gly"
    poly_pep = poly_pep + "-" + amino
    m = m + 3
    n = n + 3
else:
    print("Translation does not occur due to the absence of a stop codon.")

if valid_gene == True:
    print("Protein has a sequence of amino acids as follows: " + poly_pep)

Please enter DNA: TTTGGCGGATTTTACAGATACTAACCGGTGATTTGAAGTAAATTAGTT
DNA is TTTGGCGGATTTTACAGATACTAACCGGTGATTTGAAGTAAATTAGTT
mRNA is AAACCGCCUAAAAUGUCUAGAUUGGUCACUAAACUUAUUAUCAA
A start codon is found at 14 - 16
A stop codon is found at 31 - 34
Protein has a sequence of amino acids as follows: Met-Ser-Met-Ile-Gly-Pro-Leu-Asn-Phe-Ile

```

Enrich

As students write the code, they may realize that errors can occur in the process. This realization presents an opportunity to discuss whether protein synthesis in the body can also involve errors, especially considering that the number of nitrogenous bases in the body is far greater than in their programs. Specifically, using the programs they created, students can explore the consequences of a nitrogenous base being omitted, added, or substituted. For example, omitting one nitrogenous base (e.g., G) in the gene shown in Figure 8 can result in a different number and sequence of amino acids (c.f., Figure 9). This change can alter the shape and structure of the resulting protein. Consequently, the protein may malfunction. By examining multiple cases of altered nitrogenous bases within a gene and their effects, students may observe that these changes can result in the failure to synthesise a protein. This phenomenon is known as a mutation, which can manifest in various, often abnormal traits. Students can then find real-world examples of such mutations on the internet. This exploration can extend their understanding of protein synthesis in the abnormal cases.

Figure 9

Protein Synthesis Output After Nucleotide Change (Mutation Example)

```
Please enter DNA: TTTGGCGGATTTTACAGATACTAACCAGTGATTTGAAGTAAATTAGTT
DNA is TTTGGCGGATTTTACAGATACTAACCAGTGATTTGAAGTAAATTAGTT
mRNA is AAACCGCCUAAAAUGUCUAUGAUUGGUCACUAAACUUCAUUUAAUCAA
A start codon is found at 14 - 16
A stop codon is found at 19 - 22
Protein has a sequence of amino acids as follows: Met-Ser-Met-Ile-Gly-His
```

Evaluate

At the end of the lesson, students should be encouraged to evaluate what they have learned. When writing programs, students' code often fails to fully reflect the real process of protein synthesis in the body. This is an inherent limitation in scientific models and modelling. Therefore, in addition to identifying similarities, it is important that students discuss how the model in their programs differs from the actual process of protein synthesis. For example, during protein synthesis, certain segments of mRNA (introns) are removed, while the remaining segments (exons) join to form a gene for translation. Moreover, several tools, such as conceptual tests (Guzman & Bartlett, 2012) and learning logs (Lawsin et al., 2022), can be used to evaluate students' understanding. Students' responses to these tools can encourage discussion to deepen their understanding. With a stronger grasp of the biological process and the nature of models, students can reflect on how their programs can be more efficient—even if they diverge from the biological process. For instance, after identifying the start codon, students may decide to locate the stop codon before initiating translation.

Conclusions and Implications

This article presents an instructional activity in which computational thinking is incorporated into the teaching of protein synthesis using the 6E Learning by DeSIGN™ model of design-based learning (Burke, 2014). The instructional activity can be considered an example of STEM-integrated lessons, given that it draws on science (protein synthesis), technology (computer programming), engineering (design process), and mathematics (logical manipulation). Whereas the first three stages of the instructional activity (Engage, Explore, and

Explain) focus primarily on science learning, the fourth stage (Engineer) emphasizes computational thinking, design process, and mathematical application. The remaining two stages (Enrich and Evaluate) are devoted to extending and assessing students' learning, respectively. Although all STEM fields are integrated to varying degrees throughout the instructional activity, these stages promote reciprocal learning through which each field supports and reinforces the others. As demonstrated in this article, the proportion of STEM components is not required to be equal in an instructional activity.

Through the instructional activity, this article offers a pedagogical guideline for science teachers to implement the activity in their classrooms. In doing so, however, science teachers must be aware of this article's limitations. Specifically, it does not—and cannot—describe the instructional activity in full detail. Moreover, the instructional activity does not target all aspects of protein synthesis. For example, while the instructional activity emphasizes the biological process, it may not directly target some misconceptions possessed by students, such as confusions about the specificity of related terms. The same limitation is also featured in computational thinking. For example, while some concepts (e.g., loops and conditionals) are frequently applied throughout the coding process, other concepts (e.g., events and parallelism) may be less used. As a result of these limitations, adjustments, expansions, and additions may be necessary to make the instructional activity more suitable for particular contexts, students, and purposes. Moreover, as in teaching any topic, science teachers require a strong foundation of content knowledge.

In addition to directly applying the instruction activity, science teachers can use it as a learning resource for creating and developing their own lessons on other science topics. As demonstrated in this article, they need to analyze a scientific concept in light of computational thinking. In other words, they should consider how the concept can be abstracted and decomposed to parts and how each part can be modeled using a computational concept. Furthermore, science teachers need to choose a model of design-based learning that supports the integration of computational thinking with the teaching of that scientific concept. Such a model does not need to be the one demonstrated in this article, but it should be appropriate for the concept being taught. If possible, the developed instructional activities should be organized according to the learning progression suggested by the extant literature, so that students can learn the concepts in a progressively sophisticated manner. The more science lessons like the one presented in this article are developed, the more students are bound to learn computational thinking and design process regularly.

Research Possibilities

Although carefully designed and developed, the instructional activity presented in this article has not yet been empirically studied. Therefore, further research is needed to determine whether and how it can influence students' learning, particularly regarding protein synthesis and computational thinking. Research with students is currently underway. Because the instructional activity also has the potential to serve as a learning resource for science teachers, there is an opportunity to examine whether and how it can support their professional development. For example, given that computational thinking entails not only concepts, but also perspectives and practices, it would be valuable to investigate whether and how science teachers develop abilities, attitudes, and identities related to coding through engaging in the instructional activity. Such characteristics may increase the likelihood that science teachers,

whether in-service or preservice, integrate computational thinking into their science lessons. For those interested in the pedagogical implications and the research directions afforded by this article, the full code is provided in the Appendix for learning.

Author Contributions

LL (1): Conceptualization, Instructional Design, Writing – Original Draft Preparation, and Writing – Review and Editing. LL (2): Software, Visualization, Writing – Original Draft Preparation, and Writing – Review and Editing.

Declaration of the Use of Generative AI

During the preparation of this manuscript, the authors used ChatGPT (OpenAI, GPT-5.3) to assist with language editing and polishing of the draft. The tool was used solely to improve clarity, grammar, and overall readability. The output was subsequently reviewed by a human editor. The authors revised the text as necessary and take full responsibility for the content of the manuscript.

Ethics

Ethical approval was not applicable because this article did not involve human participants, animals, or identifiable personal data.

References

- Bacovic, M., Andrijasevic, Z., & Pejovic, B. (2022). STEM education and growth in Europe. *Journal of the Knowledge Economy*, 13(3), 2348–2371. <https://doi.org/10.1007/s13132-021-00817-7>
- Brennan, K., & Resnick, M. (2012). New frameworks for studying and assessing the development of computational thinking. *Proceedings of the 2012 Annual Meeting of the American Educational Research Association* (pp. 1-25). American Educational Research Association.
- Bureau of Academic Affairs and Educational Standards. (2017). *Indicators and core learning content in science according to the basic education core curriculum B.E. 2551 (revised version B.E. 2560)*. Press of the Agricultural Co-operative Federation of Thailand.
- Burke, B. N. (2014). The ITEEA 6E *Learning byDeSIGN™*: maximizing informed design and inquiry in the integrative STEM classroom. *Technology and Engineering Teacher*, 73(6), 14–19.
- Delen, I., & Sen, S. (2023). Effect of design-based learning on achievement in K-12 education: A meta-analysis. *Journal of Research in Science Teaching*, 60(2), 330-356. <https://doi.org/10.1002/tea.21800>
- Doppelt, Y., Mehalik, M. M., Schunn, C. D., Silk, E., & Krysinski, D. (2008). Engagement and achievements: A case study of design-based learning in a science context. *Journal of Technology Education*, 19(2), 22–39.
- Fisher, K. M. (1985). A misconception in biology: Amino acids and translation. *Journal of Research in Science Teaching*, 22(1), 53–62. <https://doi.org/10.1002/tea.3660220105>
- Grover, S. & Pea, R. (2013). Computational thinking in K–12: A review of the state of the field. *Educational Researcher*, 42(1), 38–43. <https://doi.org/10.3102/0013189X12463051>

- Guzman, K. & Bartlett, J. (2012). Using simple manipulatives to improve student comprehension of a complex biological process: Protein synthesis. *Biochemistry and Molecular Biology Education*, 40(5), 320–327. <https://doi.org/10.1002/bmb.20638>
- Hsiao, H., Lin, Y., Lin, K., Lin, C., Chen, J., & Chen, J. (2022). Using robot-based practices to develop an activity that incorporated the 6E model to improve elementary school students' learning performances. *Interactive Learning Environments*, 30(1), 85–99. <https://doi.org/10.1080/10494820.2019.1636090>
- Institute for the Promotion of Teaching Science and Technology. (2014). *Basic knowledge: STEM education*. Institute for the Promotion of Teaching Science and Technology
- Kite, V. & Park, S. (2023). What's computational thinking?: Secondary science teachers' conceptualizations of computational thinking (CT) and perceived barriers to CT integration. *Journal of Science Teacher Education*, 34(4), 391–414. <https://doi.org/10.1080/1046560X.2022.2110068>
- Koba, S. & Tweed, A. (2009). *Hard-to-teach biology concepts: a framework to deepen student understanding*. National Science Teachers Association Press.
- Ladachart, L., & Ladachart, L. (2020). Please mind the gap between science and design. *Suranaree Journal of Social Science*, 14(2), 118–132. <https://doi.org/10.55766/HLCI2091>
- Lawsin, N. L. P., Esguerra, C. T., & Garcia, G. S. (2022). Exploring student mental models and student experiences to improve the teaching practice of cell and molecular biology. *American Journal of Multidisciplinary Research and Innovation*, 1(5), 69-74. <https://doi.org/10.54536/ajmri.v1i5.826>
- Li, Y., Schoenfeld, A. H., diSessa, A. A., Graesser, A. C., Benson, L. C., English, L. D., & Duschl, R. A. (2020). On computational thinking and STEM education. *Journal of STEM Education Research*, 3(2), 147-166. <https://doi.org/10.1007/s41979-020-00044-w>
- Markowetz, F. (2017). All biology is computational biology. *PLoS Biology*, 15(3), e2002050. <https://doi.org/10.1371/journal.pbio.2002050>
- National Research Council. (2007). *Taking science to school: learning and teaching science in grades K-8*. The National Academics.
- Office of the National Economic and Social Development Council. (2022). *The thirteenth national economic and social development plan (2023–2027)*. Royal Gazette.
- Ogegbo, A. A., & Ramnarain, U. (2022). Teachers' perceptions of and concerns about integrating computational thinking into science teaching after a professional development activity. *African Journal of Research in Mathematics, Science and Technology Education*, 26(3), 181-191. <https://doi.org/10.1080/18117295.2022.2133739>
- Open Oregon. (2019). *Codon table [Image]*. Open Oregon Pressbooks.
- Peel, A. & Friedrichsen, P. (2018). Algorithms, abstractions, and iterations: teaching computational thinking using protein synthesis translation. *The American Biology Teacher*, 80(1), 21–28. <https://doi.org/10.1525/abt.2018.80.1.21>
- Pleasants, J., Tank, K. M., & Olson, J. K. (2021). Conceptual connections between science and engineering in elementary teachers' unit plans. *International Journal of STEM Education*, 8, article number 16. <https://doi.org/10.1186/s40594-021-00274-3>
- Promboon, S., Finley, F. N., & Kaweeikijmanee, K. (2018). The evolution and current status of STEM education in Thailand: Policy directions and recommendations. In G. W. Fry (Ed.), *Education in Thailand: An old elephant in search of a new mahout* (pp. 423–459). Springer. https://doi.org/10.1007/978-981-10-7857-6_17

- Qin, H. (2009). Teaching computational thinking through bioinformatics to biology students. In S. Fitzgerald, M., Guzdial, G., Lewandowski, S. Wolfman, & T. J. Cortina (Eds.), *Proceedings of the 40th ACM technical symposium on computer science education* (pp. 188-191). Association for Computing Machinery.
- Quinn, C. M., Reid, J. W., & Gardner, G. E. (2020). S + T + M = E as a convergent model for the nature of STEM. *Science and Education*, 29(4), 881–898. <https://doi.org/10.1007/s11191-020-00130-w>
- Schatz, M. C. (2012). Computational thinking in the era of big data biology. *Genome Biology*, 13(11), 177. <https://doi.org/10.1186/gb-2012-13-11-177>
- Shute, V. L., Sun, C., & Asbell-Clarke, J. (2017). Demystifying computational thinking. *Educational Research Review*, 22, 142–158. <https://doi.org/10.1016/j.edurev.2017.09.003>
- United Nations. (2015). *Science, technology and innovation policy review: Thailand*. UNCTAD.
- Wing, J. M. (2006). Computational thinking. *Communications of the ACM*, 49(3), 33–35. <https://doi.org/10.1145/1118178.1118215>
- Yadav, A., Krist, C., Good, J., & Caeli, E. N. (2018). Computational thinking in elementary classrooms: Measuring teacher understanding of computational ideas for teaching science. *Computer Science Education*, 28(4), 371-400. <https://doi.org/10.1080/08993408.2018.1560550>

Appendix

```

#DNA input and examination
DNA = str(input("Please enter DNA:"))      #Ask for an input (DNA)
DNA = DNA.upper()                        #Capitalize the input
for i in range(len(DNA)):                 #Test if the input comprises correct nitrogenous bases
    if DNA[i] == "A" or DNA[i] == "T" or DNA[i] == "C" or DNA[i] == "G":
        valid_DNA = True
    else:
        valid_DNA = False
if valid_DNA == True:                     #Display the result of testing
    print("DNA is", DNA)
else:
    print("DNA is not valid.")

#DNA transcription
mRNA = ""                                #Create an empty variable for mRNA
for j in DNA:                             #Pair complementary bases to form mRNA
    if j == "A":
        j = "U"
    elif j == "T":
        j = "A"
    elif j == "C":
        j = "G"
    elif j == "G":
        j = "C"
    mRNA = mRNA + j
print("mRNA is", mRNA)                   #Display mRNA

#Finding a start codon
startCODON = False                       #Set a variable to indicate if there is a start codon
for k in range(len(mRNA)):                #repeatedly find a start codon in mRNA base by base
    base1=mRNA[k:k + 1]
    base2=mRNA[k + 1:k + 2]
    base3=mRNA[k + 2:k + 3]
    if base1 == "A" and base2 == "U" and base3 == "G":
        startCODON = True                #Set the variable to indicate that a start codon is found
        break                             #Stop finding a start codon when it was found
    else:
        k = k+1
if startCODON == True:
    print("A start codon is found at", k + 1, "-", k + 3) #Display where the start codon was found
else:
    print("A start codon is not found.")      #Display where the start codon was not found

#Translation
gene = mRNA[k:] #Create a variable for a gene beginning with the start codon found
m = 0
n = 3
poly_pep = "Met"                          #Create a variable for a polypeptide with the amino acid for the start codon
valid_gene = False                         #Set a variable to indicate if the gene has a stop codon (i.e., gene is correct)
while n < len(gene) + 3:
    codon = gene[m + 3: n + 3]             #Create a variable for a codon to be translated

```

```

if codon == "UAA" or codon == "UAG" or codon == "UGA": #Repeatedly find a stop codon
    valid_gene = True          #Set the variable to indicate gene correctly has a stop codon
    print("A stop codon is found at", m + 4, "-", n + 4) #Display where the stop codon was found
    break                      #Stop finding a stop codon when it was found
elif codon == "UUU" or codon == "UUC":          #Translate the gene codon by codon
    amino = "Phe"
elif codon == "UUA" or codon == "UUG" or codon == "CUU" or codon == "CUC" or codon == "CUA"
or codon == "CUG":
    amino = "Leu"
elif codon == "AUU" or codon == "AUC" or codon == "AUA":
    amino = "Ile"
elif codon == "AUG":
    amino = "Met"
elif codon == "GUU" or codon == "GUC" or codon == "GUA" or codon == "GUG":
    amino = "Val"
elif codon == "UCU" or codon == "UCC" or codon == "UCA" or codon == "UCG" or codon == "AGU"
or codon == "AGC":
    amino = "Ser"
elif codon == "CCU" or codon == "CCC" or codon == "CCA" or codon == "CCG":
    amino = "Pro"
elif codon == "ACU" or codon == "ACC" or codon == "ACA" or codon == "ACG":
    amino = "Thr"
elif codon == "GCU" or codon == "GCC" or codon == "GCA" or codon == "GCG":
    amino = "Ala"
elif codon == "UAU" or codon == "UAC":
    amino = "Tyr"
elif codon == "CAU" or codon == "CAC":
    amino = "His"
elif codon == "CAA" or codon == "CAG":
    amino = "Gln"
elif codon == "AAU" or codon == "AAC":
    amino = "Asn"
elif codon == "AAA" or codon == "AAG":
    amino = "Lys"
elif codon == "GAU" or codon == "GAC":
    amino = "Asp"
elif codon == "GAA" or codon == "GAG":
    amino = "Glu"
elif codon == "UGU" or codon == "UGC":
    amino = "Cys"
elif codon == "UGG":
    amino = "Trp"
elif codon == "CGU" or codon == "CGC" or codon == "CGA" or codon == "CGG" or codon == "AGA" or
codon == "AGG":
    amino = "Arg"
elif codon == "GGU" or codon == "GGC" or codon == "GGA" or codon == "GGC":
    amino = "Gly"
poly_pep = poly_pep + "-" + amino #Add the current codon's amino acid to the polypeptide
m = m + 3
n = n + 3
else:
    print("Translation fails due to missing stop codon.") #Display when the gene has no stop codon

if valid_gene == True:
    print("Protein has a sequence of amino acids as follows: " + poly_pep) #Display the polypeptide

```