Computational Thinking

Reading & Vocabulary Development for CS50x Learners

Authored by Shabnam Shahlapour

Winter 2024

Table of Content

❑ How to Study?

- **Step 1:**

In the first section (Reading), read through the article once completely.
Don't worry about understanding every word—just focus on getting a general sense of the content.

- **Step 2:**

Next, check out the list of words and expressions from the text.
See which ones are familiar to you and which ones are new.
Some unfamiliar words can be guessed from the context, and some familiar ones might have a different meaning here than usual.

- **Step 3:**

In the following sections, you'll find explanations and example sentences for those words and phrases.
Some of them might not appear directly in the article, but since they're related, it's a good opportunity to get to know them.

This process continues throughout the material.
You can study at your own pace, take breaks whenever you need to, and come back to it later.

❑ About the Vocabulary

The vocabulary in this article isn't sorted by language level, but it generally falls into three categories:

1. High-frequency, common words that you'll remember naturally over time through repetition and exposure.

2. Specialized IT and computer terms that will become easier to learn the more you engage with tech-related content.

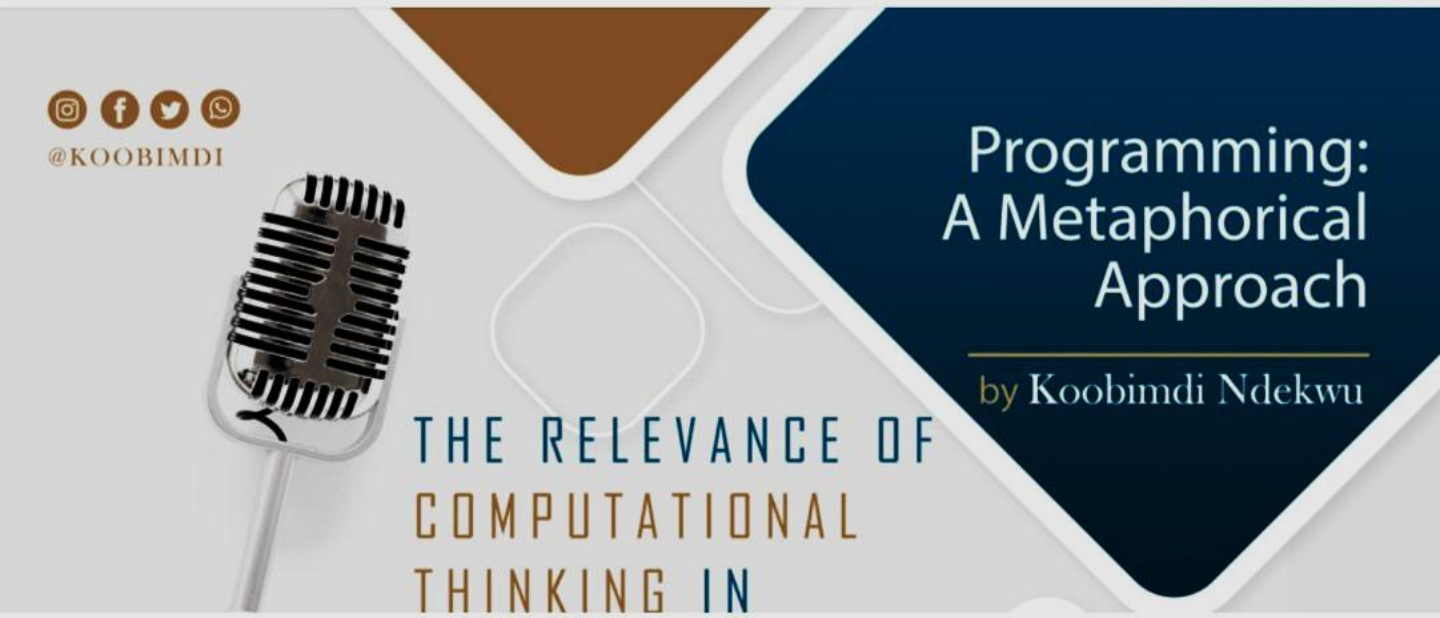
3. Less common words that you might only see in a few specific contexts.

❖ Important Note

You don't need to memorize any of these words or expressions my friend.
They're just here to support your learning journey.
The more you come across them in different contexts, the more naturally they'll stick.

[complete article link]

https://dev.to/koobimdi/the-relevance-of-computational-thinking-in-programming-cf8



Koobimdi Ndekwu
Posted on 13 Aug 2024

The Relevance of Computational Thinking in Programming

#webdev #programming #softwaredevelopment #productivity

"Why don't programmers like nature? It has too many bugs!"

Let me start by saying that Programming, in its very essence, is actually the art of solving problems. But as anyone who has dabbled into coding knows, the problems we face in software development aren't just about writing code. Most times, they're actually about thinking. It can be fairly said that, in programming, if you can get the logic and flow right, then you have solved half the problem. This is where computational thinking (CT) steps in! This is a way of thinking that's as fundamental to programming as a hammer is to a carpenter. But what exactly is computational thinking, and why is it so crucial in programming? Let me try to break it down with some metaphors, case studies, and practical examples.

What is Computational Thinking?

Think of computational thinking as the blueprint to a house you're about to build. Before you start hammering nails or laying bricks (writing code), you need a clear plan – an architectural design that outlines what you're going to build, how it will function, and how everything will fit together. Computational thinking is the mental process that helps you design that blueprint.

It involves breaking down complex problems into smaller, manageable pieces, identifying patterns, abstracting details to focus on the essentials, and developing step-by-step solutions. In other words, it's the process of "thinking like a computer" to solve problems in a logical and efficient way.

Painting a Clearer Picture using Metaphors and Case Studies

1. Decomposition (Breaking Down the Problem): Imagine you're organizing a huge dinner party. You wouldn't tackle everything at once. Instead, you'd rather break down the tasks – inviting guests, planning the menu, cooking the food, setting the table, etc. In programming, decomposition is the process of breaking down a large, complex problem into smaller, more manageable tasks.

Case Study: Suppose you're developing an e-commerce website. Instead of trying to build the entire site at once, you break it down into smaller parts: user authentication, product listing, shopping cart, payment processing, etc. Each part is a smaller problem that's easier to solve.

2. Pattern Recognition (Finding Similarities): Let's say you're baking cookies, cakes, and pies. While each recipe is different, they all share common steps – mixing ingredients, baking in the oven, etc. In programming, pattern recognition is about identifying similarities or patterns in problems that can help you apply the same solution to different situations.

Case Study: When writing code, you might notice that many functions in your program share a common pattern. For example, when processing user inputs, you might always validate the input, check for errors, and then execute some logic. Recognizing this pattern allows you to create a reusable function that handles input processing for multiple parts of your program.

3. Abstraction (Focusing on What's Important): Think of abstraction as packing for a trip. You can't take everything, so you focus on what's necessary – clothes, toiletries, maybe a book or two. In programming, abstraction is about focusing on the essential details and ignoring the irrelevant ones, making the problem easier to manage.

Case Study: When developing a game, you don't need to simulate every blade of grass in a field; instead, you focus on the broader elements like character movement, scoring, and levels. Abstraction allows you to simplify the problem by ignoring unnecessary details, making the development process more efficient.

4. Algorithm Design (Creating a Step-by-Step Solution): Imagine you're giving someone directions to your house. You wouldn't just say, "Find my house!" Instead, you'd most likely provide a step-by-step guide which will be something like "turn left at the filling station, go straight for 100 meters, and so on. In programming, algorithm design is about creating a clear, step-by-step solution to a problem. Just like Mike Ross said in *Suits*, "The law is a specific endeavour." In a strict sense, the same can be said of computer programming.

Case Study: Let's say you're developing a search feature for a website. You might design an algorithm that searches through a database of items and returns results based on user input. The algorithm might involve steps like checking each item, comparing it to the search term, and then displaying the matching results.

❑ Phrases & Vocabulary

relevance
irrelevant
in its very essence
dabble into sth
case study
endeavor
in a strict sense
simulate
every blade of grass in a field

relevance [noun, also *relevancy*]
the quality or state of being closely connected or appropriate
Opposite:
irrelevance

relevant [adj.]
connected with what is happening or being discussed; meaningfully related to sth
Opposite:
irrelevant

---

relation [noun]
- connection or similarity
- Relations are the connections between people, groups, organizations, or countries
- a person who is a member of the same family as another person

related [adj.]
- belonging to the same family
- connected
- connected to, influenced by, or caused by something

related costs/expenses
related products/services
related activities/changes/causes
be related to sth

essence

- the basic meaning or importance of something
- a strong liquid, usually from a plant or flower, that is used to add a flavor or smell to something

vanilla essence
essence of violets

dabble

If you dabble in something, you take part in it but not very seriously
dabble in
dabble with
Synonyms for “dabble in” (more common everyday):
toy with, play around with

case study

a detailed study of a person, group, or thing, especially in order to show general principles

endeavor

- [noun] an attempt to achieve a goal
- [verb] try hard to do or achieve something

Synonyms:
attempt, try, bid, effort

in a/the strict sense
in the most limited meaning of a word, phrase, etc.
Synonyms:
literally, accurately, in the true/literal sense, rigorously

blade
- the thin, flat cutting part of a tool or weapon
- a blade is also a thin, flat leaf of grass

simulate [verb]
to create conditions or processes similar to something that exists

simulated [adj.]
made to look like something else; artificial

simulator [noun]
a piece of equipment that is designed to represent real conditions, for example in an aircraft or spacecraft

❑ Exercise 1

Fill in the blanks with the words below. Some word forms may need to be changed.

essence(x2) simulation(x3) blade(x2)
strictest dabble endeavor

1. I don’t paint much, I just _____.
2. She wasn't a farmer in the _____ sense - she never actually worked in the fields.
3. Yet change is the very _____ of life.
4. In _____, both sides agree on the issue.
5. The public bombarded the company with complaints in an _____ to have the price increases revoked.
6. While walking through the meadow, I noticed a _____ of grass caught between the _____ of my pocket knife
7. During pilot training, instructors _____ various flight scenarios using advanced flight _____. These devices provide a _____ environment that closely mirrors real-life conditions, allowing trainees to practice and hone their skills safely.

Answer Key

❑ Exercise 1-2

Fill in the blanks with the words below.

relevant related relative relatively prevalent pertinent

In the information technology (IT) sector, certain trends have become increasingly _____(1). One notable development is the rise of "new-collar" jobs, which emphasize skills over traditional educational backgrounds. These roles are particularly _____(2) in today's rapidly evolving tech landscape, as they focus on competencies directly _____(3) to current industry needs. The _____(4) importance of formal degrees has diminished, with employers placing greater value on practical abilities. Consequently, individuals from _____(5) diverse educational paths are entering the IT workforce, equipped with _____(6) skills acquired through alternative learning avenues.

Answer Key

Practical Use-Case Example Scenarios

Scenario – Automated Email Sorting: Lets assume you're writing a program to automatically sort incoming emails into different folders based on keywords. You'd start with decomposition, where you'd break down the task into scanning the subject line, identifying keywords, and moving the email to the appropriate folder. You'd use pattern recognition to identify similar keywords across different emails, abstraction to focus on the key details (subject line and keywords), and algorithm design to create a step-by-step process for sorting.

Scenario 2 – Building a Weather App: If you're developing a weather app, you might decompose the project into fetching data from an API, processing the data, and displaying it to the user. Pattern recognition helps you identify common tasks like data fetching that can be reused. Abstraction allows you to focus on the essential data (temperature, humidity, etc.), and algorithm design helps you create a process for updating the app with new weather information.

Key Points to Take Note Of

- CT is Fundamental: Computational thinking isn't just for programmers, it's a problem-solving skill applicable across various fields.
- Decompose Complex Problems: Break down large problems into smaller, more manageable tasks.
- Recognize Patterns: Identify similarities across different problems to apply common solutions.
- Focus on Essentials: Use abstraction to manage complexity by focusing on what's important.
- Design Clear Algorithms: Develop step-by-step solutions to problems.

Conclusion

Computational thinking is the backbone of effective programming. It's the difference between hacking together code that might work and designing a solution that's robust, efficient, and scalable. By mastering computational thinking, you equip yourself with the skills to tackle any programming challenge with confidence and clarity.

"Remember, in programming, it's not just about thinking outside the box . . . sometimes, it's actually about thinking inside a well-structured function!"

❑ Phrases & Vocabulary

sort sth into sth
based on
pattern recognition

fetch data from an API
applicable across various fields
break down into
equip yourself with
backbone
hack
tackle

assume: take for granted, suppose ,imagine
apply: use, utilize, employ, implement
robust: strong, tough

fetch (get)
to go get something or someone and bring the thing or person back

- ## What is Data Fetching?

Fetch is the retrieval of data by a software program, script, or hardware device. After being retrieved, the data is moved to an alternate location or displayed on a screen.

backbone

- (body part) your spine
- (important part) the part of something that provides strength and support
- (character) strength of character or bravery
- (IT) the system of equipment and connections that allows communication at high speeds over long distances

Have some backbone!
Show a little backbone!
She has no backbone (at all).
has enough backbone to [win through, succeed, try]
is the backbone of the [company, manufacturing sector, nation]
the [company's] backbone
the [moral, social] backbone of this [company]
[forms, makes up] the backbone of society
[broke, shattered, severed] her backbone

hack

- (Unauthorized Access) Gaining illegal entry into computer systems or networks.
- (Creative Solution) A clever, quick or unconventional fix to a problem.
- (Recreational Programming) Experimenting with technology or coding for enjoyment.
- (Cough) A short, dry cough.

tackle (deal with)

to try to deal with something or someone

tackle [a problem, crime, issues]
tackle the [difficult, tricky, thorny] problem of
tackle the (mountain of) debt

❑ Exercise 2

Fill in the blanks with the words below. Some word forms may need to be changed.

fetch backbone(x2) tackle(x2) robust(x2)

1. Data centers form the _____ of the internet and the banking system.
2. There are many ways of _____ this problem.
3. I was determined to _____ my boss on the way I had been treated.
4. Could you _____ the kids from school?
5. He looks _____ and healthy enough.
6. Exports will continue to be bolstered by the _____ economy.
7. The delegates had enough _____ to reject the proposal.

Answer Key

1 back to exercise

1. dabble
2. strictest
3. essence
4. essence
5. endeavor
6. blade-blades
7. simulate-simulators-simulated

1-2 back to exercise

1. prevalent
2. relevant
3. pertinent
4. relative
5. relatively
6. related

2 back to exercise

1. backbone
2. tackling
3. tackle
4. fetch
5. robust
6. robust
7. backbone

Sources

Cambridge, Oxford, Merriam-Webster, Collins Dictionaries
Ludwig.guru
Thesaurus.com

DEV community
IGI Global

CS50x Iran

Harvard's Computer Science 50x Iran