The term Computational Thinking refers to the human approach to understanding complex problems, so that a solution may be designed that could be understood by both and computers.

There are four fundamental features of Computational Thinking that help us to tackle these problems.

1. Decomposition – breaking the problem down into simpler and more manageable parts
2. Pattern recognition – finding similarities in problems so we can apply similar logic
3. Abstraction – focusing on what we know and understand and ignoring irrelevant information
4. Algorithms – producing step-by-step instructions that can be programmed to solve the problem

[1]

I will use a practical example to explain Computation Thinking and demonstrate its relevance and importance to future learning on this course and into my future career using an example problem from CMT 120.

A screen shot of a computer program

Description automatically generated*“Write code that prints out the numbers 1 - 1000, but if the number is divisible by 3 it prints ‘fizz’, if the number is divisible by 5 it prints ‘buzz’ and if the number is divisible by 3 and 5 it prints ‘fizzbuzz’”*

**Decomposition**

Before trying to solve a large and complex problem it can be useful to break that problem down into smaller and simpler parts. In this example we can identify at least 4 components of the problem statement:

1. Printing out the numbers 1 – 1000
2. Printing out the word ‘*fizz’* in place of numbers that are divisible by 3
3. Printing out the word ‘*buzz’* in place of numbers that are divisible by 5
4. Printing out the word ‘*fizzbuzz’* in place of numbers that are divisible by both 3 and 5

**Pattern recognition**

Problems 2 – 4 are very similar in their natural language wording and as we begin to write the code to solve these problems, we recognise that they share a similar pattern.

1. Printing a word in place of a number
2. Only printing the word if a certain condition is met (divisibility by 3, 5 or 3 & 5)

**Abstraction**

By focussing only on the relevant important information, we can save ourselves from becoming sidetracked by information we do not need or confused by information we do not understand.

Using an else (else console.log(i);) clause to finish the if, else if, else conditional block is a simple method of logging all numbers that are neither divisible by 3 or 5 without considering any other factors.

**Algorithms**

By decomposing the problem into its components, recognising the pattern that certain problems share and abstracting away from any unimportant details, we can formulate a concise step-by-step instruction to solve the problem.

We must change the order of the instructions to get our program working correctly. Problem component 4 must come before 2 and 3 or we will never produce fizzbuzz as the iteration’s divisibility by 3 will be first tested and will be true, resulting in fizz in place of fizzbuzz.

**Conclusion**

Computational Thinking has already proved to be a useful technique in solving-problems encountered within this course. Moving forward with the course and into my future career, I am sure to encounter more complex problems which will require a methodical approach. Utilising the four cornerstones of Computational Thinking will allow me to simplify, understand and solve increasingly more complex problems.

**References:**

[1] BBC. 2023, [What is computational thinking? - Introduction to computational thinking - KS3 Computer Science Revision - BBC Bitesize](https://www.bbc.co.uk/bitesize/guides/zp92mp3/revision/1)

Q 2.

**Ray Tomlinson**

**<img>**

Born April 23, 1941

Amsterdam, New York US

Died: March 5, 2016 (74)

Lincoln, Massachusetts

Ray Tomlinson was a Computer Engineer best-known for his work in the field of computer networking and email communication.

In 1971, while working at Bolt, Beranek and Newman under contract to the United States Department of Defense, Tomlinson was working on reprograming SNDMSG to work on an operating system called TENEX. SNDMSG was a program that allowed users to send messages locally to other users on a shared computer. Tomlinson had also been working on a separate program called CPYNET which allowed file transfer through the budding ARPANET.

Tomlinson took code from CPYNET and combined it with code from the SNDMSG program to send the first email – a message from one user, sent to another user on a separate computer. Tomlinson chose the @ sign to append user and host names and thus the user@host email address format was born. [1]

Some say the choice of the @ sign was arbitrary, however Tomlinson was quoted saying that the @ symbol “just makes sense” as it “indicate a unit price (for example, 10 items @ $1.95). I used the at sign to indicate that the user was “at” some other host rather than being local.” [2]

IT took a couple of decades for Tomlinson’s invention to become anything more than a novelty experiment. But by 1996, with the increased accessibility of personal computers and the demand for faster communication in both business and social lives, email had become more commonly used than postage mail in the U.S. [1]

Tomlinson’s contribution to Computer Science and Business endures today. Email accelerated the development of the internet in its earlier stages through its utility as a tool for the dissemination of information among other internet users and developers. Email set the stage for the internet to be used as a platform for knowledge and information exchange as well as a personal communication tool. This can be seen in the enduring utility of email within our personal, professional and academic lives. Email also paved the way for later forms of internet communication such as instant messaging and social media.

Whether we like it or not, email has transformed the way we work, access information and communicate with one another and despite its downsides, we have yet to devise a better system. In a world where an app’s popularity can rise and fall in a matter of months and businesses start-up and fold on a daily basis; emails longevity is a testament to Tomlinson’s achievements and his contribution not only to the field of Computer Science, but also to business, communication, edication, journalism and much more. [3][4]

[1] Lemelson-MIT. [Ray Tomlinson | Lemelson (mit.edu)](https://lemelson.mit.edu/resources/ray-tomlinson)

[2] [Robert Mackay, The New York Times. 2009. Internet Star @ Least 473 Years Old - The New York Times (nytimes.com)](https://archive.nytimes.com/thelede.blogs.nytimes.com/2009/05/04/internet-star-least-473-years-old/)

[3] Eric Barton, BBC. 2015 [Love it or loathe it, email changed the world - BBC Worklife](https://www.bbc.com/worklife/article/20150109-love-it-or-loathe-it-email-changed-the-world)

[4] Dave Lee, BBC. 2016 [Ray Tomlinson's email is flawed, but never bettered - BBC News](https://www.bbc.co.uk/news/technology-35742210)

* Programmer who implemented the first email protocol on the ARPANET system
* Credited with the invention of the TCP three-way handshake which underlies the HTTP and many other key Internet protocols

**Referencing!!**

**Use [1], [2], [3], hyperlink style referencing**

**COOOOOOLERs**

3.

A short (400-500 word) reflection on what you have learnt in this module, and how that will impact on your learning for the rest of your course

On this course, we’ve covered a number of interesting and formative topics including:

* The fundamentals of Computational Thinking
* History of the internet
* HTML & CSS
* Basic Javascript
* Team projects

These topics will help us moving forward both throughout this course and into our professional careers beyond this course as they include some of the building blocks of required to be a successful IT Professional.

Learning the fundamentals of Computational Thinking has equipped me with the knowledge of how to approach complex problems, break them down into their parts, simplify them and generate solutions. This will be invaluable in all aspects of this course and my future career as I will continually be faced with increasingly more challenging and complex problems which will require a Computational Thinking approach to solve.

We’ve learned about the history of the internet and how it has developed and evolved over the years.

We’ve learned to write HTML & CSS to build websites and Javascript to solve programming problems. I’ve often found myself unsure of how to solve a problem and had to conduct further research into finding the solution by using resources such as MDN, W3School and Stack Overflow. This aspect of continuous research and learning is something that every IT professional must do. We can’t know everything, but understanding the logic behind problem solving and being able to find the missing syntax or command word is essential in developing the solution and being a successful programmer.

One of the most formative aspects of the module, in my opinion, was the task we were set to produce a simple website as a team. This forced us to communicate plans and ideas, assign responsibilities, share progress, blend our ideas and code into one website and present this to the rest of the module group. We faced challenges with a lack of clear direction to begin with, so looking at example websites and designing a basic wireframe helped us clarify this. We also struggled with finding a good way to share our progress. We opted to use a shared Google Doc for this, and while this allowed us to share our code, it became a large and messy document that was far from optimal for the situation. We later learned about Github and decided this would have been a much more suitable option for code sharing and version control. Given that our CMT 133 (Software Engineering) module is almost entirely groupwork focused, I’m glad that we had the chance to work in a team and identify some areas that did and didn’t work before taking on a bigger project.

The key skills developed in this module share similarities with upcoming and current modules and the lessons learnt and mistakes made along the way are already proving to be useful. I’m confident that as I progress further through the course and into my career in tech, I will continue to be able to apply what I learned in this module.

|  |  |
| --- | --- |
| **WHAT I’VE LEARNED** | **HOW IT WILL IMPACT ON LEARNING** |
| CT | All modules  Described in Q1 – Building blocks for problem simplification and solving. |
| History of internet | All modules – background knowledge – internet came from a series of projects and its current state is far from the original plans. Iterations will generate new use cases and public demand will drive new iterations and functionalities |
| HTML & CSS | Fundamentals of programming – building basic websites  Learning the basics early on will give us a good foundation when implementing mor ecomplex functionality within our sites using JS.  We’ve learned a lot about how to structure and adapt our sites according to display sizes.  HTML – basic elements of webpages  CSS – styling of webpages   * User centric programming * Device oriented programming |
| Basic Javascript | Fundamentals of programming  Gaining an introduction into the syntax of this language will help us to solve programming problems in FoP and give us a basis to build upon for more complex web design |
| Teamwork | Software Engineering  We ran a mini team project to build a basic site. This will help |

The four components identified in our Decomposition are all correct, however, written in such order they will produce an error. If we first command our program to write ‘*fizz’* in place of all numbers that are divisible by 3, when we reach our first number that is divisible by both 3 and 5, the condition of 3 divisibility will be tested first and it wield be true and therefore fizz will be logged.

Therefore, we must first test for divisibility by both 3 & 5. Giving us the Algorithm in the order seen in this image.

Using an if, else if, else conditional and a modulus operator with the number we aretesting for their divisibility, followed by a console.log() command. We can

By breaking a large and complex problem into smaller, more manageable parts, we are able to simplify the problems and more clearly understand what steps it will take to solve the problem as a whole.

In the context of this project, I was able to decompose the complex problem of creating a static HTML & CSS website and simplify this into it’s component problems. Using the pro-forma and considering my own preferences, I identified the criteria as such:

* Create a static HTML & CSS website
  + Homepage to link other pages
    - Header/responsive nav bar
      * Media queries
      * Hamburger Menu
    - Links in flex boxes
    - Footer
      * Add social media links
      * Flex and centre
  + Page for task 1, 2 and 3
    - Header/responsive nav bar
      * Media queries
      * Hamburger Menu
    - Content section
      * Task answer
      * Use a grid layout for content and picture alignment
    - Footer
      * Add social media links
      * Flex and centre

This technique allows me to visulise the component problems of the more complex problem and solve them individually. This is something that I will be using often in the future throughout this course and future career.

*This is a crucial part of any problem-solving technique and will be highly important as I progress through this course and my career.*

*Within this project I decomposed the “problem” of building a website by first familiarising myself with the marking criteria, identifying the problem components (3 separate tasks, content = 60% of grading, HTML & CSS = 30% grading etc.) and addressing them individually to simplify the process.*

*A Software Developer or Business Analyst will use decomposition techniques when planning to build a product by breaking down the products into its components, the features of its components, their specifications and so on. This is done until a clear image of the tasks needed to be completed is formed.*

**Pattern Recognition**

If we find that certain features of the product/program we are building share similarities and functionalities with other features, we may be able to reuse code we have already written and save ourselves time and effort.

In the example of this project, I understood that the header and footer should have the same styling on each page. The task pages should also share styling for the “main” section. Therefore, I can tag the HTML elements appropriately and use the same CSS styling sheet to render the same effects within these pages.

**Abstraction**

By focussing only on the relevant important information, we can save ourselves from becoming sidetracked by information we do not need or confused by information we do not understand.

**Algorithms**

Once we have understood the problem, recognised the patterns we can write code that follows a step-by-step process to solve these problems.