Computational thinking files for the workshop	
Workshop Powerpoint file	What I use for teaching – amend away
2024-05-20_comp-think-open-version.pptx	Editable ppt file
Counts of names	
counts.txt	Output file from loop created by allfiles.sh
2023-11-20_tolkien_counts.xlsx	Formatted version of output file
Individual loop scripts	
king.sh	Name counts for *The Return of the King*
fellow.sh	Name counts for *The Fellowship of the Ring*
two.sh	Name counts for *The Two Towers*
Combined script	
allfiles.sh	Name counts for all three texts in one go
Texts for loop exercise	
fellowship.txt	Text file of *The Fellowship of the Ring*
return.txt	Text file of *The Return of the King*
towers.txt	Text file of *The Two Towers*

All files downloadable from here:

https://github.com/weaverbel/intro-computational-thinking

Slide 1 and 2

Introduce the aims and learning objectives of the workshop. Explain that no prior knowledge is required.

Slide 3

Give a definition. Explain that computational thinking is useful even in daily life – to plan complex activities, manage projects.

Slides 4 and 5

Explain the difference between the thinking and the doing. Computational thinking helps define what tasks need to be done. Programming will allow those tasks to be coded.

Slide 6

Introduce the complex problem to be used during the workshop. Obviously, this is not a computable problem, but the methodology of breaking down a complex problem is the same regardless. It is ideal once people understand the methodology to have them try to solve the problem in small groups.

Slides 7 and 8

The building blocks of computational thinking. Explain that there are these four steps. An example will follow to help flesh this out.

Slide 9

While this mathematical example is more a trick than anything, it does provide an easy and concrete example of the methodology of computational thinking.

Slide 10

Spend quite a bit of time making sure people ‘get’ the steps, because the final step – abstraction – is very important – being able to RE-USE generalised steps for similar problems.

Slide 11

Let them try to apply the same steps to different numbers. This practice helps them ‘get’ it. Don’t muddy the waters by getting them to work with odd numbers as that is a different set of steps.

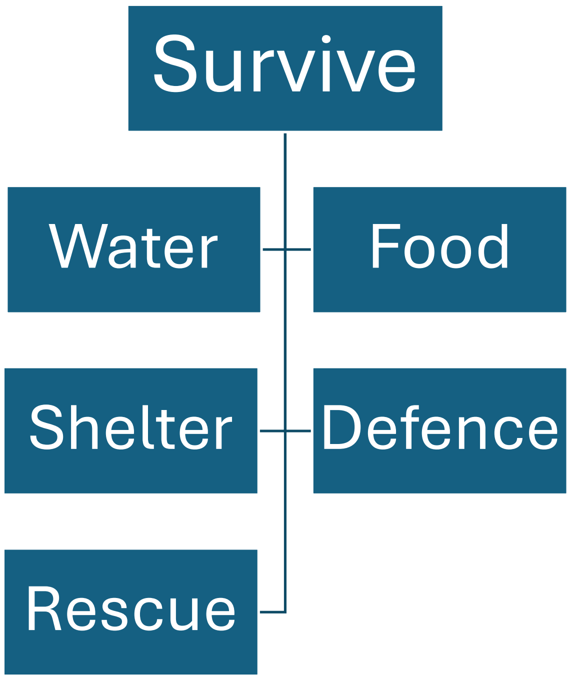
Slide 12

Structure diagrams are used in project management and other kinds of planning to break problems down. Explain that the MAIN objective, the KEY objective, should go at the top. Then underneath, there will be other subheadings, and then subheadings under each of the subheadings and so on until the lowest level of individual tasks is reached.

Slide 13

This is the place for small groups to work together to try to agree on the key objective, then the subheadings and the tasks.

e.g.



The above is an example of top level concerns under SURVIVE but each subcategory will need further breakdown and further tasks, e.g.,

Water	Shelter	Food	Defence	Rescue
• Locate • Vessels to carry & store • Map sites	• Design • Materials • Tools • Fire for warmth • Sanitation	• Investigate plant and animal sources • Means to cook (fire)	• Weapons • Perimeter fence	• Signal fire • Lettering on beach •

See that fire appears in both rescue and shelter and would also appear under food because of the need to cook. Tasks under each of these key needs would need to be broken down further.

Explain that the structure diagrams help a group come to a consensus about key objectives and the tasks to achieve it. From my experience, groups had quite different views about the key objective, which then affected the sub-tasks.

Slide 14

This is an example of how to break down a task into easily computable steps. There would be plenty of other ways to do this task – this is just one example of how it might be achieved. Explain that it might seem like a lot of effort, but that once written, the computer will execute these coded steps in a heartbeat. The code written to achieve this can then be generalised to do the same task or a similar task on a different text or set of texts.

Slide 15

Explain that code rarely moves in this kind of linear way. Rather, code takes different paths depending on conditions. Don't go into too much detail here – just walk them through the example onscreen so they understand the basics of branching in coding, the `if/else` that makes code adaptable to different conditions.

Slide 16

Give them another go at structure diagrams.

Slide 17

Explain pseudocode.

Pseudocode is a description of the steps that a computer program will perform, without needing to follow the syntax rules of any particular programming language. It's the intermediate step between planning a piece of code and coding it in a specific language.

Pseudocode helps people conceptualise the algorithmic steps of a solution before writing it in any particular language.

It's often used in the early stages of software development to ensure software developers agree on the tasks, and can explain those tasks to a non-coder, i.e. a manager.

Switch to the shell and execute one of the individual loop scripts - make sure you have the appropriate `.txt` files in the directory - `return.txt`, `towers.txt`, and `fellowship.txt` before launching any script.

- `king.sh`
- `fellow.sh`
- `two.sh`

Explain that the code to do the task can be easily changed to do the same task on a different text. Run all three scripts individually, then run the combined script, `allfiles.sh`, to get counts across all three texts in one go – explain that code can constantly be tweaked to be faster and more efficient. The file `counts.txt` can be printed to the screen to show them the counts per character.

Print the code to a screen and talk people through what each of the components mean. Don’t go into too much detail – just enough to point out that you are executing a loop on the text of a certain book.

Slide 18

This just gives an example of a loop which they saw executed before.

Slide 19

This exercise helps people write pseudocode for an actual task they might want to do. From my experience teaching humanities people, this exercise was beyond most of them, so if that is the case, try to get them to use structure diagrams to break the problem down.

Slide 20

This exercise helps people write pseudocode for a more complex task they might want to do.

From my experience teaching humanities people, this exercise was beyond most of them, so if that is the case, try to get them to use structure diagrams to break the problem down.

Slide 21

Recap the steps of computational thinking and ask for questions and comments.

Get feedback via sticky notes.