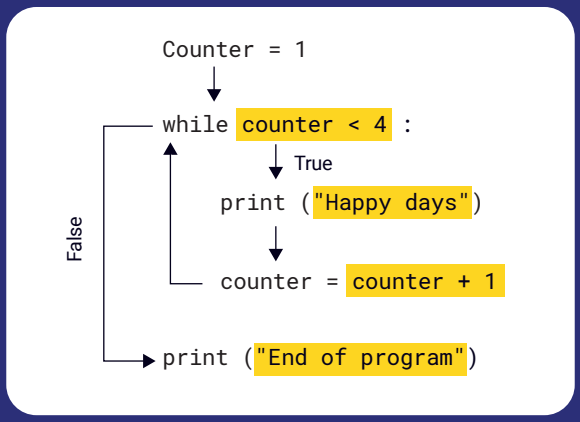
Raspberry Pi Foundation

Pedagogy Quick Read

Code tracing

Developed in the early 2000s, code tracing is a well-established approach to help learners develop their program comprehension. Put simply, it involves reading and analysing code, before running it to predict its outcome. Novice programmers should be competent in code tracing before they can confidently write programs of their own.



1. Highlight all the expressions
2. Use arrows to show the order of execution
3. Follow the program and fill in the variables table and the output box

Summary

Tracing involves:

- Reading the code
- Interpreting the meaning
- Recording the flow and/or outputs

Benefits of tracing:

- Fosters program comprehension
- Improves code writing
- Supports learners to be able to analyse and explain code
- Exposes misconceptions
- Reduces cognitive load
- Helps learners develop a consistent notional machine

Variables table

counter	
1	
2	
3	
4	

What is output?

```
Happy days
Happy days
Happy days
End of program
```

▲ In the example above, elements of the program flow and code are provided. By completing the variables table, learners can demonstrate their understanding of the program through code tracing.

What is code tracing?

It is widely understood that young learners should ideally have developed some reading skills before they begin learning to write. Similarly, in computing, there is a body of evidence to suggest that code tracing, a form of code reading, is an effective precursor to code writing and independent programming.

When tracing code, learners review chunks of code, or whole programs, and record its expected behaviour and execution flow at various stages. This can be captured through annotation as well as recording the program output at each stage. The Teach Computing Curriculum 'Programming' units include examples of code tracing tasks. Learners may be asked to **trace a piece of code**, predict the outcome, and then be guided through the code, line by line to test their prediction. Typically, prediction is done away from the computer to ensure learners focus on reading rather than executing the code. Learners could also be given short sections of code in the form of **worked examples**, or complete trace tables, where some values are provided and learners use code tracing to record the missing values. With this secure understanding, learners can then be given the opportunity to create their own program featuring the concepts they have traced. While there is no single approach to tracing, there are some clearly defined methods such as TRACS[1] which may be useful for learners to follow.

Here, we explore the benefits of code tracing, how it fits in with the concept of the notional machine, strategies to employ to lighten cognitive load, and how code tracing can be used in the classroom.

Benefits?

Harrington identified that when learning programming, learners build their understanding in a hierarchical way, with tracing at the most basic level, then explaining the code, before they progress on to writing. Many other studies have been completed based on this theory. Hertz and Jump, who developed the 'trace-based-teaching' model, found that starting a class with 20 to 30 minutes of tracing increased attainment and decreased drop-out rates.[2] A 2004 study found that learners who could trace effectively less than 50% of the time could also not explain it effectively.[3] If we accept there is a broad consensus advocating code tracing as an effective strategy with a broad range of evidence to support the claim, what should we consider when using it in the classroom?

The notional machine

To trace code effectively, learners must have some understanding of the notional machine. This concept was first introduced by Benedict du Boulay and describes the conceptual model that learners have about how a computer processes instructions and data.[4]

The notional machine can look very different depending on the type of programming language being used. In Scratch, it is simple to run more than one process concurrently (threading), whereas in most text-based languages (including Python) this is more complex. This has implications for how we begin to teach programming in Scratch. Learners may demonstrate that they can use threads but may not understand how the machine handles them. This gap in their notional machine understanding can lead to gaps in their knowledge or to misconceptions. Encouraging learners to use threads in Scratch without addressing the notional machine may lead to later problems for teachers when learners find threading more difficult to achieve in Python.

Lightening the load

Code tracing can contribute to a reduction of the cognitive load placed on learners.

In focusing learners' efforts on existing and working programs, and answering specific questions, educators can avoid unnecessary extraneous load being placed on their learners. By giving learners the opportunity to trace code, they can comprehend the code and its function before they see it in action. This approach also helps develop their understanding of the notional machine — how the code is executed. Many other areas of the curriculum explore similar ideas, such as Talk for Writing in literacy and progressing from concrete objects to abstract numerals in mathematics.

In context application

Code tracing can be incorporated in the classroom as a stand-alone acitivity or as part of a wider approach.

- The PRIMM (Predict, Run, Investigate, Modify, Make) [5] approach is ideally suited to it as it required learners to Predict in its first step, which involves reading and tracing.
- Begin a programming activity or project by providing learners with an existing project or snippet of code.
- Tracing is also a good way to check learners' understanding of the capabilities of the notional machine. Using examples where specific misconceptions may lead to an incorrect solution, the act of tracing can expose and help address this misconception.

References

1. Donaldson, P. and Cutts, Q., 2018, October. Flexible low-cost activities to develop novice code comprehension skills in schools. In *Proceedings of the 13th Workshop in Primary and Secondary Computing Education* (pp. 1-4).

2. Hertz, M. and Jump, M., 2013, March. Trace-based teaching in early programming courses. In *Proceeding of the 44th ACM Technical Symposium on Computer Science Education* (pp. 561-566).

3. Lister, R., Adams, E.S., Fitzgerald, S., Fone, W., Hamer, J., Lindholm, M., McCartney, R., Moström, J.E., Sanders, K., Seppälä, O. and Simon, B., 2004. A multi-national study of reading and tracing skills in novice programmers. *ACM SIGCSE Bulletin, 36*(4), pp. 119-150.

4. Du Boulay, B., 1986. Some difficulties of learning to program. *Journal of Educational Computing Research, 2*(1), pp. 57-73.

5. Raspberry Pi Foundation, *Pedagogy Quick Reads: Using PRIMM to structure programming lessons.* Available from: the-cc.io/qr11

Raspberry Pi Foundation