Raspberry Pi Foundation

Pedagogy Quick Read

Improving program comprehension through Parson's Problems

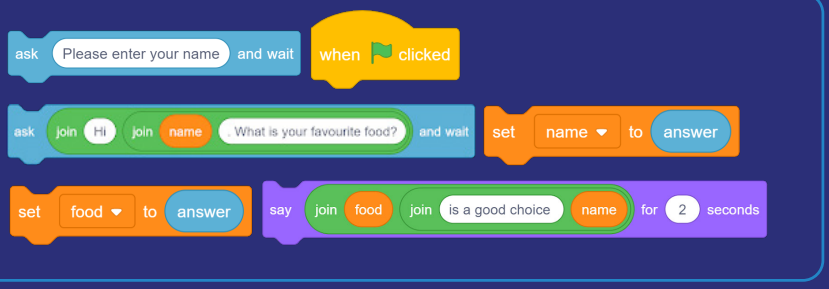
An important precursor to learning to write computer programs is having the necessary **program comprehension** to interpret the function and structure of existing programs. One tool that can help learners develop program comprehension is Parson's Problems, which are exercises that require learners to rearrange lines of code into the correct sequence.

Rearrange the lines or blocks of code below to create a program that asks the user for their name, then for their favourite food, before telling them that their food choice is a good choice.

Python

```
print("Hi "+ name + ".   What is your favourite food?")
print(food  +  "  is a good choice "  + name)
food  =  input()
name  =  input()
print("Please enter your name:  ")
```

Scratch



What is a Parson's Problem?

A Parson's Problem is a task in which learners are given all of the blocks or lines of code needed to solve a problem, however, the lines have been jumbled so that they are no longer in the correct order. Learners are asked to reorganise the code into the correct order to perform a specific task.

The short example above shows some jumbled lines of code (in Python and Scratch), and sets out the task that needs to be completed. Why not see if you can solve the problems in the example?

Parson's Problems can be applied to both text- and block-based programming and can vary in difficulty, to accommodate learners' existing understanding. For example, when you feel that learners are ready, they could be provided with lines of code and be expected to work out the indentation themselves (known as 2D Parson's Problems).

There are many ways in which Parson's Problems can be presented to learners. They make for an excellent offline or paper-based activity that could be done individually, in pairs, or in small groups. You may choose to create problems directly in the development environment to allow learners to immediately test their solutions. Alternatively, there are online tools such as **js-parsons** that allow you to create your own interactive problems.

Parson's Problems can be used to support formative assessment, as classroom discussion following the activity plays an important part in learners' development. Immediate feedback also avoids any misconceptions being committed to long-term memory.

Summary

Parson's Problems reduce the cognitive load for learners, reducing the need to recall syntax; instead, learners can focus on program structure and logic in a way that is low-stakes and engaging.

Parson's problems support learners by:

- Developing learners' understanding of how the program is executed (notional machine)
- Reducing cognitive load
- Focusing on blocks of code rather than syntax
- Providing all the correct code within an engaging challenge
- Promoting dialogue and discussion about code

Benefits of Parson's Problems:

- Constrain the logic
- Avoid common syntax errors that can be barriers to learning to code
- Model good programming practices
- Provide the potential for immediate feedback
- Make it easier to identify common misconceptions
- Increase engagement of learners

Advice for writing Parson's Problems:

- Share problems with only a single solution
- Allow learners to manipulate actual code blocks
- Provide a clear description of the problem
- Clearly show the desired logic
- Share multiple similar problems over time

The benefits of Parson's Problems

The main benefit of Parson's Problems is that the learner is focusing on the structure and logic of blocks of code, rather than the syntax of individual text elements (atoms). The process reduces the **cognitive load** experienced by learners, allowing them to practise sequencing and problem-solving with code. This experience is particularly helpful in the early stages of learning to program, when learners may be easily frustrated and put off by repeated unsuccessful attempts to solve a problem. Parson's Problems also expose learners to logic and syntax that they may not be fully familiar with.

Denny et al.[1] suggest that learners' solutions to a Parson's Problem "make clear what students don't know (specifically in both syntax and logic)". These solutions can allow for an easier analysis of the common errors that learners make, whereas "the open-ended nature of code-writing questions makes identifying such errors difficult". For example, when using a Parson's Problem, we can be sure that an error was not caused by a typing mistake.

Parson's Problems can promote some higher-order thinking in learners than simple code tracing (reading code and identifying its purpose or output). Parson's Problems can act as a stepping stone between the lowest and highest categories — being able to read and interpret code and being able to write original code, which involves evaluation and creation (the highest categories in Bloom's).

Izu et al.[2] place Parson's Problems in the 'Blocks' row of the **Block Model** proposed by Schulte. They state that "novice programmers should develop program comprehension skills as they learn to code so that they are able both to read and reason about code created by others, and to reflect on their code when writing, debugging or extending it". They also state that Parson's Problems support learners in developing their understanding of the notional machine.

Distractors

Some Parson's Problems include distractors. Distractors are incorrect blocks or lines of code that are included in the set of provided code, meaning that learners need to be selective about which blocks they use.

Rearrange the lines of code to create a program that outputs the total cost to the customer. Be aware that there are two lines of code that will cause errors in your program if used.

```
price = 3.50
quantity = 5
total = price * Quantity
total = price * quantity
print(total)
print("total")
```

The inclusion of distractors can add an additional level of challenge[3] for more confident learners. However, care should be taken, as they may unnecessarily increase the cognitive load or the time spent on a task, or even result in a misconception or error being committed to long-term memory.

Advice for writing Parson's Problems

Provide learners with a clear explanation of what the program should do when correctly sequenced — doing so reduces their cognitive load. Additionally, Denny et al.[1] recommend making sure that there is a unique answer for each question, ie there should only be one order of the lines that achieves the goal.

Ensure that learners manipulate the actual lines of code, rather than using letters or numbers as a shorthand. Working with real lines of code helps to develop their familiarity with the syntax and the construction of the code.

In theory, it is possible for learners to guess the correct answer to a simple Parson's Problem without fully understanding the construct or logic being tested. Asking more than one question over time that tests the same logic or construct can reduce this concern.

Providing structure (eg braces, colons, indentation) can make a question more accessible, as learners can use these visual clues to develop their solution. Providing this structure can also make problems including more complex programming concepts possible.

References

1. Denny, P., Luxton-Reilly, A. & Simon, B. (2008) Evaluating a New Exam Question: Parsons Problems. In: *ICER '08: Proceedings of the Fourth International Workshop on Computing Education Research*. New York, ACM. pp. 113–124.

2. Izu, C., Schulte, C., Aggarwal, A., Cutts, Q., Duran, R., Gutica, M., Heinemann, B., Kraemer, E., Lonati, V., Mirolo, C. & Weeda, R. (2019) Fostering Program Comprehension in Novice Programmers - Learning Activities and Learning Trajectories. In: *ITiCSE-WGR '19: Proceedings of the Working Group Reports on Innovation and Technology in Computer Science Education*. New York, ACM. pp. 27–52.

3. Harms, K. J., Chen, J. & Kelleher, C. L. (2016) Distractors in Parsons Problems Decrease Learning Efficiency for Young Novice Programmers. In: *ICER '16: Proceedings of the 2016 ACM Conference on International Computing Education Research*. New York, ACM. pp. 241–250.

Raspberry Pi Foundation