Python 10: 1D Lists

Teachers Resource

Resources

- Slides **Python 10: 1D Lists**
- You will need to either have Python IDLE installed or have access to an online Python IDE. We have used https://editor.raspberrypi.org/en
- Activity worksheets are included in this lesson. You will need to distribute these to your pupils
- We have added a walk-through video below

Prior Knowledge

Printing, mathematical operators, data types and input using Python

- Data types in Python
- counted `while` loops in Python
- Selection using `if..elif..else` in a `while` loop

3MB

Python 10 -1D Lists.pptx

Lesson walk-through

You need to log in to access videos

Learning Objectives

To teach pupils to:

- be able to use indexes to reference individual elements is a list
- be able to use a `while` loop to iterate over a list
- be able to use selection in a `while` loop lists
- be able to use parallel lists

Teacher Note

It is a deliberate decision to use `while` loops before `for` loops. Iterating is a fundamental operation in programming, and using a `while` loop for iteration can help pupils to understand the concept of manual index handling and loop control. Once this is properly understood and pupils can use it with more complex problems, then `for` loops are introduced as a useful and efficient shortcut.

Slide Notes

Starter

Slides 2 - 3: Show this code and give pupils time to rearrange the lines. There are many ways of tackling this, you could:

- print the code out and cut it up so they can physically rearrange the lines
- you could type the code into an online Parson's Puzzle tool such as https://parsons.problemsolving.io
- you could ask pupils to write down the line numbers and note which lines are indented

The solution is shown on slide 3.

Slide 4: Remind pupils about the use of square brackets in strings. If your class are feeling less confident as you hoped, you might like to ask them to try this code.

Slide 5: Show this slide and describe the use of lists. You might like to precede this slide with a discussion about how you could store the names of all the pupils in the class. You would not want to have individual variables as that would be difficult to manipulate, and you would have to edit your code if you had new students. What we would like is a way of storing multiple values in the same variable. You could use a single string and put spaces in between, but that wouldn't work with some names, and it would take a lot of processing.

Link the fact that a string is a sequence of elements, which are single characters, and a list is a sequence of elements that can be entire strings, integers etc.

Highlight the fact that we use indexes and square brackets in the same way as we did with strings.

Activity 1 - Iterating over a list

Slide 7: Show the code which iterates over the names list and highlight how a list is created in code and how you can access a single element.

Pupils should then type the code in and run it. It is preferable that they type it in rather than copy, as they will pay more attention to it if they have to type it. You might like to ask them to change the code, for example, they could add more elements in the list, only output every second element or iterate backwards.

They should then go on to the exercises. The solutions follow on slides 9 - 10.

Slide 9: Check that pupils have used `len(food)` and not just used a fixed value. Some pupils will erroneously type this to get the last element.

```
print(f"The last item is {food[length]}")
```

Remind pupils that indexes start at 0, so the last element is at index position `length - 1`.

Slide 10: We have not explicitly told pupils how to change an element in a list, so they might need a hint. See if they can make a sensible guess and they can try it out.

Activity 2: List Operations

Slide 12: append() adds to the end of a list. Point out the syntax, as it is common for pupils to misuse this command.

Slide 13: remove() deletes a specific element from a list. You provide the value you want to remove, not the index.

Slide 14: Ask pupils to try the `append` and `remove` examples and then do just Exercise 1 on their worksheet.

Slide 15: This slide shows the solution and includes the output if the colour they want to remove actually exists in the list.

Slide 16: This slide shows the error message received when we try to remove something that doesn't exist in a list. Ask pupils what they should do to avoid their program crashing like this. They can't be sure that the user will enter the values correctly, so they must change their code.

Slide 17: Use this slide to support your discussion about how to avoid a program crashing if the value you are trying to remove is not in the list.

Slide 18: This slide shows a loop which iterates over the colours list and remembers if an input colour is found using a Boolean variable, found. Give pupils time to read this code. You might find it beneficial to ask them to type it in and run it. Once they are confident with this, move on to the next slide.

Slide 19: This slide shows how we can use the Boolean variable, `found`, to only remove the input element if it was found in the list.

Slides 20–21: This slide shows how we can output an error message if the input value is not found in the list. Pupils should try this code and then complete the exercises.

Slide 22: This is the solution to Exercise 2.

Slide 23: The solution to Exercise 3 is shown on this slide. It extends the idea of checking if something is found in the list before doing something else.

Slide 24: This slide shows the output when the user enters a colour that was already in the list.

Activity 3: Parallel Lists

Slide 26: Ask pupils to predict what this code will output. Do not allow them to run it until they have made their prediction. It doesn't matter if they are wrong; the important thing is that they think about it.

Slide 27: Once they have an answer, ask them to run the code. You could then continue the PRIMM approach by encouraging them to investigate the code, make changes to it, and use it as a model for a different exercise.

Slides 29–31: These slides show the answers to the parallel lists exercise.

Plenary

Pupils should discuss the code shown and consider why you must not sort parallel lists. Notice that there is nothing that links the `names` list to the `scores` list, we must keep them in the same order.