Python 9: String Manipulation

Teaching Resource

Resources

- Slides **Python 9: String Manipulation**
- 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

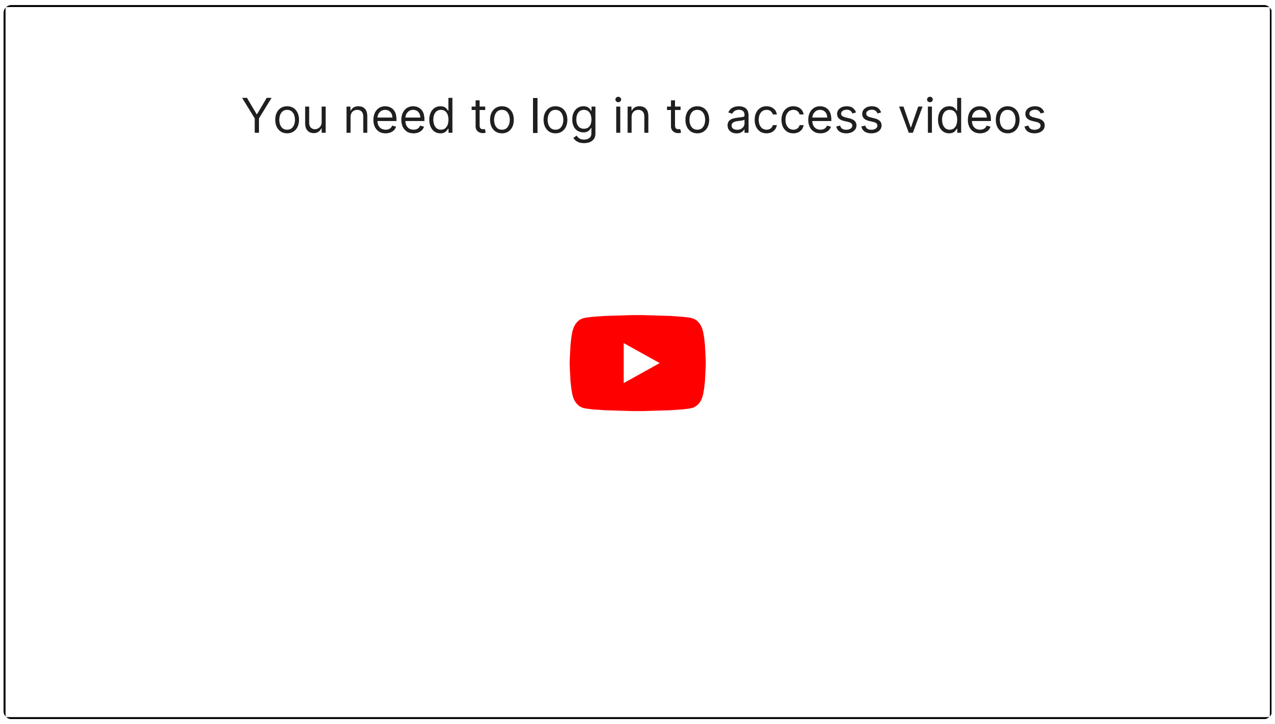
Prior Knowledge

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 9 - String manipulation.pptx

Lesson walk-through


You need to log in to access videos

Learning Objectives

To teach pupils to:

- recognise when a variable is a string
- be able to use indexes to reference individual characters is a string
- be able to use a `while` loop to iterate over a string
- be able to use selection in a `while` loop and nested `while` loops with strings

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 the code and ask pupils to work out which of the variables shown are strings. They should answer on their worksheets. The answers are shown on slide 3. It is common for pupils to think that `"123"` is not a string because it only contains digits. However, it is enclosed in quotes, so it is a string. Equally, `True` is not a string, it is a Boolean, because it is not in quotes.

Activity 1: String Functions

Slide 5: Show this slide and discuss how we can use the `len` function to find out how many characters are in a string. Notice that the `len` function returns a value which must either be used immediately, or stored in a variable. I.e.

```
name = input("What is your name? ")
# You can use the returned value immediately
print("The length is", len(name))

# Or you can store it in a variable to use later
length = len(name)
print("The length is", length)
```

Note: it is important to use correct terminology such as using 'function' here, even without going into what a function is in programming terms.

Slide 6: Discuss how we can use the `.upper()` or `.lower()` functions to get an uppercase or lowercase version of a string.

Common error: it is very common for pupils to forget the round brackets. This can be quite confusing as sometimes it just appears that nothing is happening or, as in the case that follows, you get confusing error messages.

```
# This is incorrect and gives this error:
# <built-in method upper of str object>
print(name.upper)

# This is correct
print(name.upper())
```

Slide 7: Notice that the `.upper()` or `.lower()` functions use dot notation, i.e. you have a variable followed by a dot and the function name and brackets, whereas the `len` function does not. The reason for this is that the `.upper()` or `.lower()` functions are specific to strings, whereas the `len` function can be used with lists. This will be covered in a later unit.

Slide 8: Ask pupils to do the worksheet exercises. Answers follow on slides 9 - 11.

Activity 2: String Formatting

Slide 13: It is common to use comma-separators for simple print statements in Python, but f-strings are useful when the output is more complex.

Slide 14: This slides shows some example f-strings. You must put a lowercase 'f' before the opening quote of a string, and then you surround variable names with curly braces.

Slide 15: Note how you can put an expression inside the curly braces, e.g. `18 - age` or `len(name)`. This is particularly useful when you only need to output the result of an expression and not really need to store it for later use. Exercise solutions follow on slide 16.

Activity 3: Sequences of Characters

Slide 18: it is important to note that strings can be broken down into individual characters. You cannot do this with other data types, such as integers and Booleans. E.g., with the string "Hello there!", we can refer to the first character using index 0. The Strings Investigation should be used to encourage pupils to find out about string indexes by observation. The answers follow on slide 19.

Activity 4: Iterating over a string

Slide 21: This slides shows how we can print individual characters in a string, e.g. "Holly", using indexes. "Holly" has five letters in it, so the indexes are 0 to 4. Pupils should be able to recognise that they would need to add further print statements with larger index values if a longer name is entered. This idea is explored on slide 22. Some might suggest that a loop should be used. Others might question what would happen if a shorter name was entered, and the answer is that the provided code would crash, as it did in the Strings Investigation task.

Slides 23–24: In these slides, we are working towards using a while loop to print out all the characters in an input strings, regardless of its length. Start by asking pupils to identify the first and last index values to be used with a given input. They should identify that the index should start at 0 and end with 4, which is `length - 1.` Then show pupils how we could use a variable for the index and increase it by 1 each time. The duplication is deliberate; you want pupils to identify a pattern which can be repeated using a loop.

Slides 25–26: This slide shows a partial solution. Pupils should complete the gaps to finish the code. The answer follows on slide 26. Once you have shown and discussed the solution, pupils should do the exercises. The exercise answers are on slides 27–29.

Activity 5: Selection in a Loop

Slide 31: Pupils should do the exercises. Answers follow on slides 32–36.

Slide 32: This slide shows a solution to the exercise that does not used nested loops. The nested loop solution is explored on slides 33–34. Note that on lines 8 and 9, it is essential that `letter ==` is repeated for each choice. This is because every part of a selection statement separated by Boolean operators must be a Boolean expression, therefore you must compare `letter` to something. It is not sufficient to use natural language phrasing here, e.g. `if letter == "a" or "e"` etc.

Slide 33: Some pupils might recognise that line 8, that continues onto line 9 is very long and could be replaced with a loop. If they do not suggest this, ask how the code would change if we wanted to count all lowercase alphabetic characters. For more able groups, encourage them to try to improve the code themselves. The answer follows on slide 34.

Slide 34: This slide shows a solution using a nested loop. Go through this code slowly and allow pupils to try it themselves before moving on.

Slide 35: This slide shows the solution to the extension, where pupils just needed to convert the letter to lowercase before comparing it, thereby counting both upper- and lower-case vowels.

Slide 36: This exercise can be tricky for many pupils. The most common error is thinking that the if statement shown on lines 12–15 can go inside the loop. It is true that lines 12-13 could go inside the loop, but then the message would be output every time we found a match, which is not what was required. However, it is not possible to know that a letter has **not** been found until you have checked the entire input phrase, therefore the print statement must be outside the loop.

This solution uses a Boolean variable called `found`, which is initially set to `False`. If a match is found, it is set to `True`. If line 9 is never reached, then we know that the letter was not found in the phrase. This Boolean value is used to control which output is given on lines 12–15.

If your class struggle with this concept, run the program given but move lines 12–15 to after line 10, inside the loop and demonstrate the issue.

Plenary

Pupils should discuss the code shown and consider why `v_index` must be set to 0 inside the outer (index) loop. The answer is that each time around the `while index < length` loop, we want to start checking the vowels from `index` position 0 again, so we must reset `v_index` to `0`.