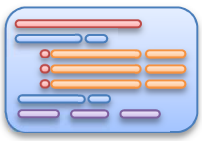


As was the case when sorting items in a list, there are a number of ways to search for a specific piece of data. We'll start with the built-in functions then look at some of the possible algorithms that might be running in the background.

Aim: *To search through items in a list for a given string.*

Task 1 – Searching with the Built-In Functions

Python has a variety of built-in functions to find data. There might be a certain word we are searching for or we might need to retrieve the data that occupies a particular position in a list.

a. Create the code on the right and save as '**15.1A Search**'.

b. Add comments explaining what each section does. Show your understanding in the output by printing lines such as "The index of the word 'learning' is 2".

Note: *The double equals sign (==) means 'is equal to'.*

```
1  words_list = ("We", "love", "learning", "to", "program")
2
3  #First
4  if "love" in words_list:
5    print("Word found!")
6
7
8  #Second
9  my_index = words_list.index("learning")
10 print(my_index)
11
12
13 #Third
14 for elem in words_list:
15   if len(elem) == 7:
16     print(elem)
17
18
19 #Fourth
20 for elem in words_list:
21   if elem[0] == "t":
22     print(words_list.index(elem))
23
24 #Fifth
25 #(slice has the same rules as with strings)
26 elem = words_list[:3]
27 print(elem)
28
29 #Sixth
30 elem = words_list[-2:-1]
31 print(elem)
```

Output

```
Word found!
2
program
3
('We', 'love', 'learning')
('to',)
>
```

c. Create a program where you can enter a word into the console and it will search for it in the list. If the program finds the word, it will say "Word found!" and tell you which number word it is in the list. Save as '**15.1B Search**'.

d. Add to the program so that if the word is not found, it says "Word missing!".

Extension

At the moment, your search is case sensitive. Searching for the word "TO" will not find the word "to". Adapt your program so that it is not case sensitive. Test the line below. It should find if the word is present (ignoring the case) but it doesn't help you find the index.

```
if your_word.lower() in map(str.lower, words_list):
```

An alternative method creates a new, lowercase version of the original list first. You can then work with this.

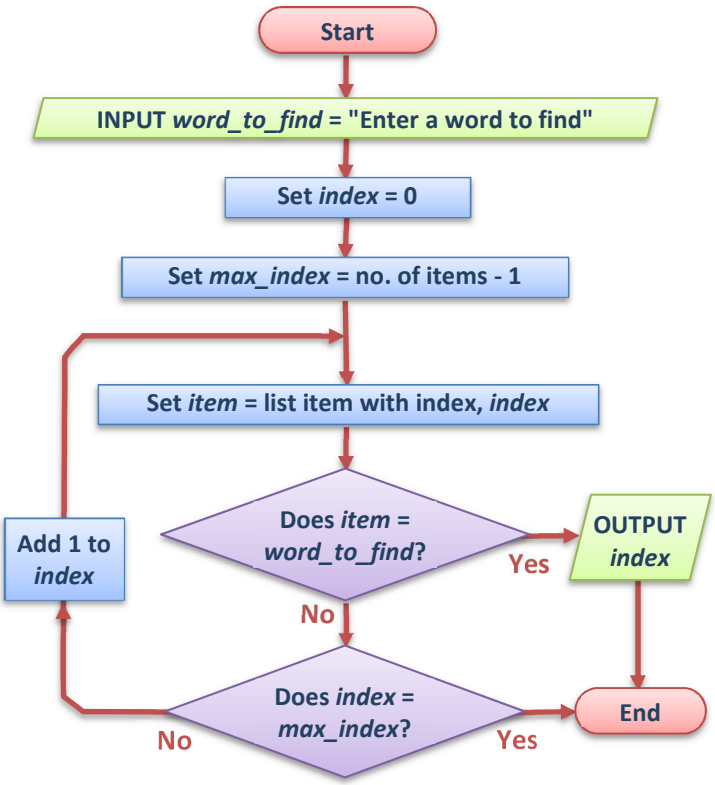
```
new_list = [item.lower() for item in words_list]
```

Searching Lists (page 2)

Task 2 – Serial Search

Serial searches look through every row in a list for the data we are searching for. The data does not need to be in order because we are looking at every item in the list anyway (at least until we have found what we are searching for).

a. The flowchart below illustrates a simple serial search. Write some pseudocode for this algorithm. You can assume that the list has already been created – just build the search function shown.



b. Write a program in *repl.it* to search through the list for the word entered using a serial search. Save as '**15.2 Serial Search**'. Here are some suggestions:

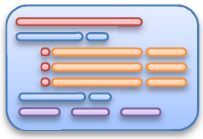
- Start by creating a list of words to search through.
- One potentially tricky part to this project is that you should exit the program once the item has been found. If you need a stepping stone to this solution, then try ignoring this requirement to start off with so that the program works through to the end of the list even if the item is found.
- To exit the program, you could use a 'while' loop with a Boolean variable. Make this variable *False* to begin with and change it to *True* if the item is found. The condition for your *while* loop might then look something like:

```
while is_word_found == False and index <= max_index:
```

Notes: *<= means 'is less than or equal to'.*

The 'and' used here is called a logical operator. Both conditions must be met for the loop to run. You could instead investigate the code `break`*, which can remove the control flow from the loop.*

Searching Lists (page 3)



Suggestions Continued

- Add the following to your program:
  - An output which states the word and its position in the list if it's found.
  - An output which states the word has not been found, if it hasn't.
  - Test output to track the variables as the process is taking place.
  - Comments explaining what is happening at each stage.

- Try and add single quotes around the search word in the output, like this:

  'Sponge' has not been found.

  You will need something like the code below. Look carefully at the quotes.

```
print( "'" + word_to_find + "' has not been found")
```

Extension: Try and include double quotes in your output instead of single ones. You can use either single or double quotes for strings in Python, so try switching these in the line shown above.

The word "Sponge" has not been found.

Task 3 – Binary Search

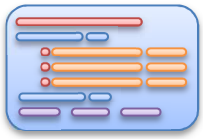
Serial searches can be slow, especially if there are thousands (or even millions) of list items to look through. *Binary searches* can be much faster but they only work with ordered lists. A binary search will look at the halfway point in a list and if that doesn't happen to be the search word, decide whether it must be above or below in the list. The unwanted half of the list is then discarded and the process repeated until the word is found.

a. The pseudocode below illustrates the basics of a binary search. Draw a flowchart for this algorithm.

If you like a challenge, add more details to your flowchart showing how you will locate the midpoint of the list and what will happen if the word is not found. Assume you know the length of the list.

INPUT *word_to_find* = "Enter a word to find"
SET *is_word_found* = False
WHILE *is_word_found* = False
  Locate midpoint of list
  IF *word_to_find* = word at midpoint of list THEN
    is_word_found = True
  ELSE IF *word_to_find* comes before midpoint THEN
    discard the second half of the list
  ELSE
    discard the first half of the list
OUTPUT index

Searching Lists (page 4)



b. Write a program in *repl.it* to perform a binary search for a word in a list. Save as '**15.3 Binary Search**'. Use the suggestions below as a guide:

- Create a list of about 10 words to search through so that you can analyse what is happening. The list needs to be in alphabetical order. Words beginning with a, b, c etc. are easy to track.
- You can reuse much of the code from the serial search program.
- You will be narrowing down the list as the program runs, discarding any unwanted sections. Introduce the variables *max_index* and *min_index* as the upper and lower boundaries of the list you haven't yet discarded.
- *min_index* will start off as 0 but will increase if the search word is not found in the first half of the list.
- *max_index* will start off as the number of items minus 1 (i.e. 9) but will decrease if the word is not found in the second half of the list.
- The halfway point in the remaining section of the list can be found at any time using the code:

```
index = int(min_index + max_index) / 2
```

  where *index* is the index of the list position that the program is currently checking.

- Think about what happens if this calculation results in a decimal number.
- Part of our solution is shown on the right. This is where the unwanted section of the list is discarded. If the word hasn't been found, either the *max_index* or *min_index* variable is changed. The *elif* statement means 'else if' in Python.

```
elif word_to_find < words_list[index]:
    max_index = index - 1
else:
    min_index = index + 1
```

- Add code to track the variables as the program is running. A simple analysis is shown on the right, but yours could show much more detail.

```
Enter a search word:  Jujube
Inspecting index 4
Inspecting index 7
Inspecting index 8
Inspecting index 9
Jujube is word no 10 in the list
>
```

- Add comments to your code to explain how it works.
- Do some analysis on the number of iterations required to find the word you are searching for compared with the serial search. What if you are searching for the first word in the list? What about the thousandth word in a list of 1000 words? Or the millionth in a list of 1 million?

Extension

There are lots of ways you could develop your binary search. Try some of the following:

- Make your binary search case sensitive.
- A binary search needs a list to be in order. Add code so that an unordered list is sorted first.
- Allow the user to input a set of words first, which would then be searched through by a second user.
- What happens if a word is present more than once? Adapt for this.
- Can you search for part of a word?