Linear search algorithm - Solution

Task 1 . An inefficient linear search algorithm

An implementation of a linear search in Python is shown in **Figure 1**. Read through the code to familiarise yourself with it - don’t worry if you don’t understand all of it yet.

| 1  2  3  4  5  6  7  8 | def linear\_search(items, search\_item):  # Initialise the variables  index = -1  current = 0  # Repeat while the end of the list has not been reached  while current < len(items):  # Compare the current item to the item you are searching for  if items[current] == search\_item:  index = current  # Proceed to the next item in the list  current = current + 1  return index |
| --- | --- |

**Figure 1**

**State** the line number where iteration is first used in **Figure 1**.

| 4 |
| --- |

**Identify** one list used in **Figure 1**.

| items |
| --- |

**Describe** what happens when line 7 is omitted from the algorithm in **Figure 1**.

| The value of current will stay at 0. This means that the condition of the while loop will always be true when the items list contains more than 0 elements, which will result in the loop running infinitely. |
| --- |

**Explain** why index needs to be initialised in **Figure 1**.

| The variable index needs to be initialised otherwise there would be an error on line 8 if the search item was not found as index would not have been defined. |
| --- |

**Explain** why the algorithm in **Figure 1** is a function and not a procedure.

| A function returns a value whilst a procedure does not. This algorithm returns a value on line 13. |
| --- |

**Complete** the trace table below using the algorithm in **Figure 1** when items is the list [“Reg”, “Chloe”, “Steph”, “Ahmed”, “Keira”, “Neelu”] and search\_item is “Keira”.

The first two passes have been completed for you.

| Line | index | current | items[current] | **Condition** |
| --- | --- | --- | --- | --- |
| 2 | -1 |  |  |  |
| 3 |  | 0 |  |  |
| 4 |  |  |  | True |
| 5 |  |  | Reg | False |
| 7 |  | 1 |  |  |
| 4 |  |  |  | True |
| 5 |  |  | Chloe | False |
| 7 |  | 2 |  |  |
| 4 |  |  |  | True |
| 5 |  |  | Steph | False |
| 7 |  | 3 |  |  |
| 4 |  |  |  | True |
| 5 |  |  | Ahmed | False |
| 7 |  | 4 |  |  |
| 4 |  |  |  | True |
| 5 |  |  | Keira | True |
| 6 | 4 |  |  |  |
| 7 |  | 5 |  |  |
| 4 |  |  |  | True |
| 5 |  |  | Neelu | False |
| 7 |  | 6 |  |  |
| 4 |  |  |  | False |

Task 2 . A more efficient linear search algorithm

**Figure 2** is a more efficient version of a linear search to the one shown in **Figure 1**.

| 1  2  3  4  5  6  7  8  9  10 | def linear\_search(items, search\_item):  # Initialise the variables  index = -1  current = 0  found = False  # Repeat while the end of the list has not been reached  # and the search item has not been found  while current < len(items) and found == False:  # Compare the current item to the item you are searching for  if items[current] == search\_item:  index = current  found = True  # Proceed to the next item in the list  current = current + 1  return index |
| --- | --- |

**Figure 2**

**State** the data type of the variable found in **Figure 2**.

| Boolean |
| --- |

The identifier found is a better choice for this variable than f. **Give** one reason why.

| The identifier found is much more meaningful than the identifier f as it signifies the purpose of the variable, which is a flag to check if the search item has been found. |
| --- |

**State** one advantage of the algorithm in **Figure 2** compared to the algorithm in **Figure 1**.

| The while loop in Figure 2 stops repeating if the search item has been found. This can reduce the number of comparisons that the algorithm needs to make and increases the efficiency. |
| --- |

**Describe** what it means if the function in **Figure 2** returns a value of -1.

| The initial value -1 signifies the item has not been found. |
| --- |

**State** three advantages of implementing the algorithm in **Figure 2** as a subroutine.

| * The subroutine can be used over and over again without repeating code. * The subroutine can be used in different programs that require searching. * The code is easier to read, maintain and debug. |
| --- |