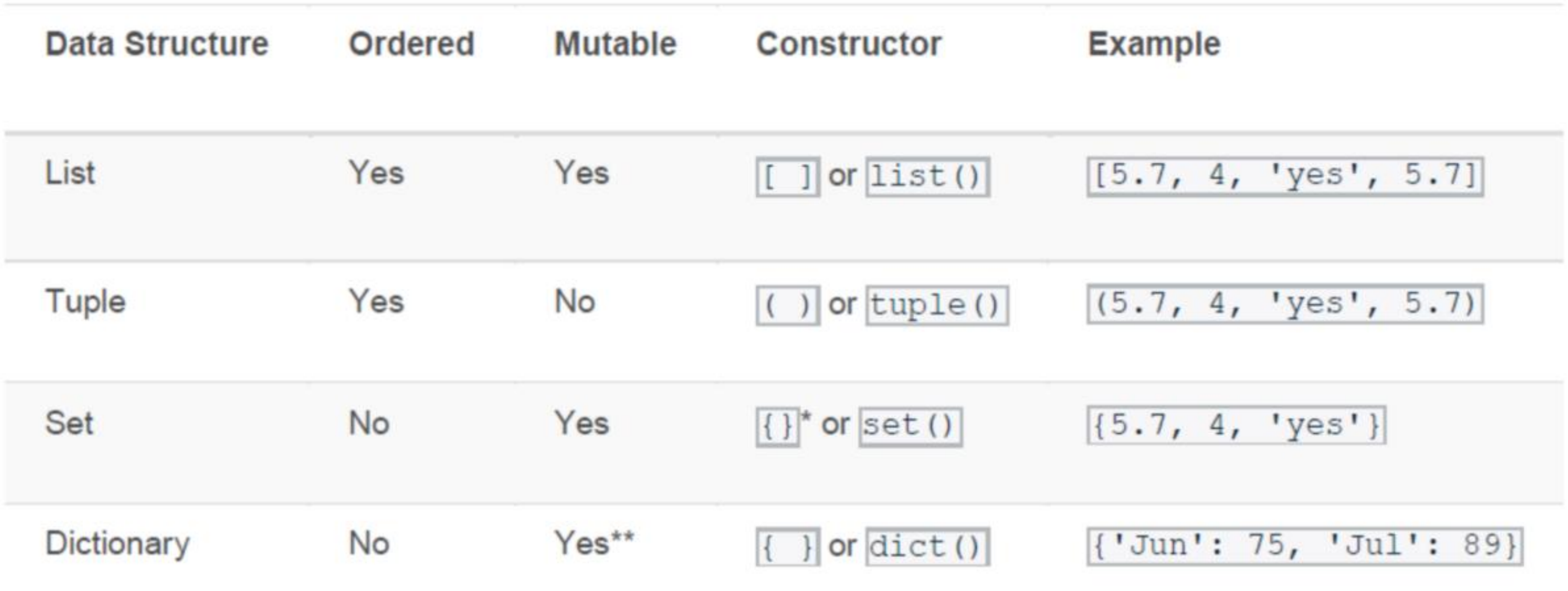
A Gentle Introduction to Python

Python Collections/Sequences (Arrays)

- **List** is a collection which is ordered and changeable. Allows duplicate members.
- **Tuple** is a collection which is ordered and unchangeable. Allows duplicate members.
- **Set** is a collection which is unordered and unindexed. No duplicate members.
- **Dictionary** is a collection which is unordered, changeable and indexed. No duplicate members.

- ***Mutable Data Types:*** Data types in python where the value assigned to a variable can be changed
- ***Immutable Data Types:*** Data types in python where the value assigned to a variable cannot be changed



Data Structure	Ordered	Mutable	Constructor	Example
List	Yes	Yes	`[ ]` or `list()`	`[5.7, 4, 'yes', 5.7]`
Tuple	Yes	No	`( )` or `tuple()`	`(5.7, 4, 'yes', 5.7)`
Set	No	Yes	`{}`* or `set()`	`{5.7, 4, 'yes'}`
Dictionary	No	Yes**	`{ }` or `dict()`	`{'Jun': 75, 'Jul': 89}`

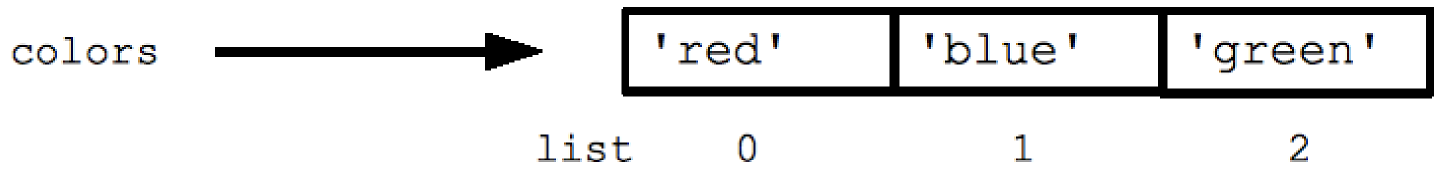
Lists

Python has a great built-in list type named "list".

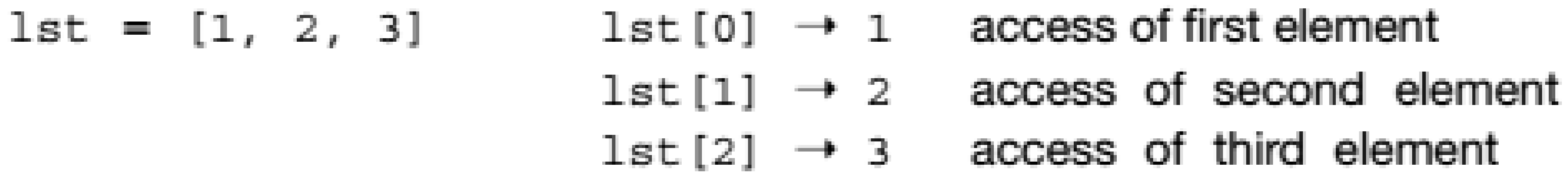
List literals are written within square brackets [ ].

Lists work similarly to strings -- use the len() function and square brackets [ ] to access data, with the first element at index 0.

```
colors = ['red', 'blue', 'green']
print(colors[0])    ## red
print(colors[2])    ## green
print(len(colors))  ## 3
```



- Elements are indexed from 0 to n-1 , where n is number of elements in the list.
- **Example:** [1, 2, 3], ['one', 'two', 'three’], ['apples', 50, True]



```
lst = [1, 2, 3]        lst[0] → 1    access of first element
                       lst[1] → 2    access of second element
                       lst[2] → 3    access of third element
```

- An empty list is denoted by an empty pair of square brackets, [].
- Negative indexing, *i.e.*, -1 and -2 refers to the last and second last items. Ex. lst[-1]=3, lst[-2] = 2.
- Range Search : lst[0:2] = [1, 2]

List Methods

Some Common List Methods:

- **list.append(elem)** -- adds a single element to the end of the list. Common error: does not return the new list, just modifies the original.
- **list.insert(index, elem)** -- inserts the element at the given index, shifting elements to the right.
- **list.extend(list2)** adds the elements in list2 to the end of the list. Using + or += on a list is similar to using extend().
- **list.index(elem)** -- searches for the given element from the start of the list and returns its index. Throws a ValueError if the element does not appear (use "in" to check without a ValueError).

List Methods

Some Common List Methods:

- **list.remove(elem)** -- searches for the first instance of the given element and removes it (throws ValueError if not present)
- **list.sort()** -- sorts the list in place (does not return it). (The sorted() function shown later is preferred.)
- **list.reverse()** -- reverses the list in place (does not return it)
- **list.pop(index)** -- removes and returns the element at the given index. Returns the rightmost element if index is omitted (roughly the opposite of append()).

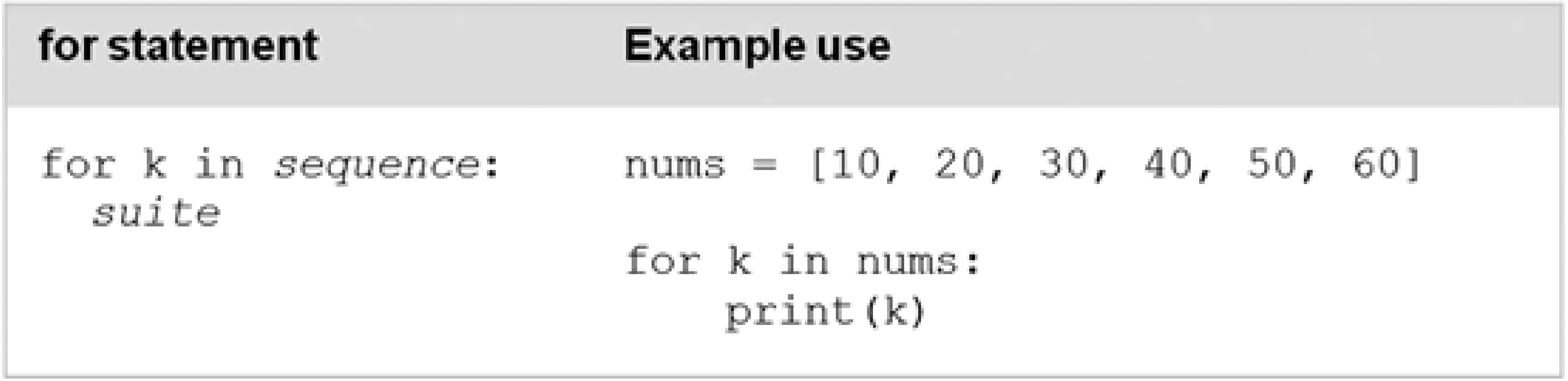
List Modification Operations in Python

Operation	fruit = ['banana', ‘apple, 'cherry']	
Replace	fruit[2] = 'coconut'	['banana', 'apple', 'coconut']
Delete	del fruit[1]	['banana', 'cherry']
Insert	fruit.insert(2, 'pear')	['banana', 'apple', 'pear', 'cherry']
Append	fruit.append('peach')	['banana', 'apple', 'cherry', 'peach']
Sort	fruit.sort()	['apple', 'banana', 'cherry']
Reverse	fruit.reverse()	['cherry', 'banana', 'apple']

Iterating Over Lists

- Python’s for statement provides a convenient means of iterating over lists (and other sequences).

- There are three ways we can do it.
  - For loops
  - While loops



for statement	Example use

`for k in` *`sequence`*`:`
    `suite`
	`nums = [10, 20, 30, 40, 50, 60]`
	
	`for k in nums:`
	    `print(k)`

Iterating Over Lists using for loops

- A **for statement** is an iterative control statement that iterates once for each element in a specified sequence of elements.

- Variable k is referred to as a **loop variable**. Since there are six elements in the provided list, the for loop iterates exactly six times.

Iterating Over Lists using while loop

- The iteration is provided as a while loop

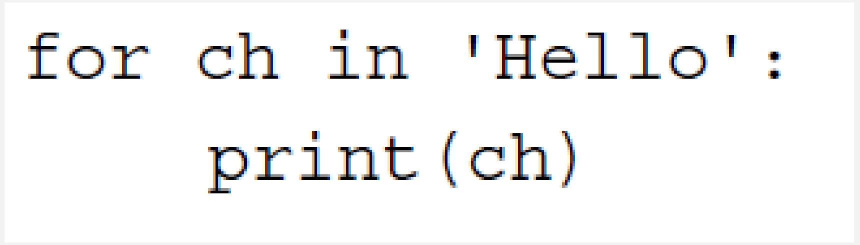
```
k = 0
while k < len(nums):
    print(nums[k])
    k = k + 1
```

- In the while loop version, loop variable k must be initialized to 0 and incremented by 1 each time through the loop.
- In the for loop version, loop variable k automatically iterates over the provided sequence of values.

Use of for loop in iteration

- The for statement can be applied to all sequence types, including strings.

- Thus, iteration over a string can be done as follows (which prints each letter on a separate line).



```
for ch in 'Hello':
    print(ch)
```

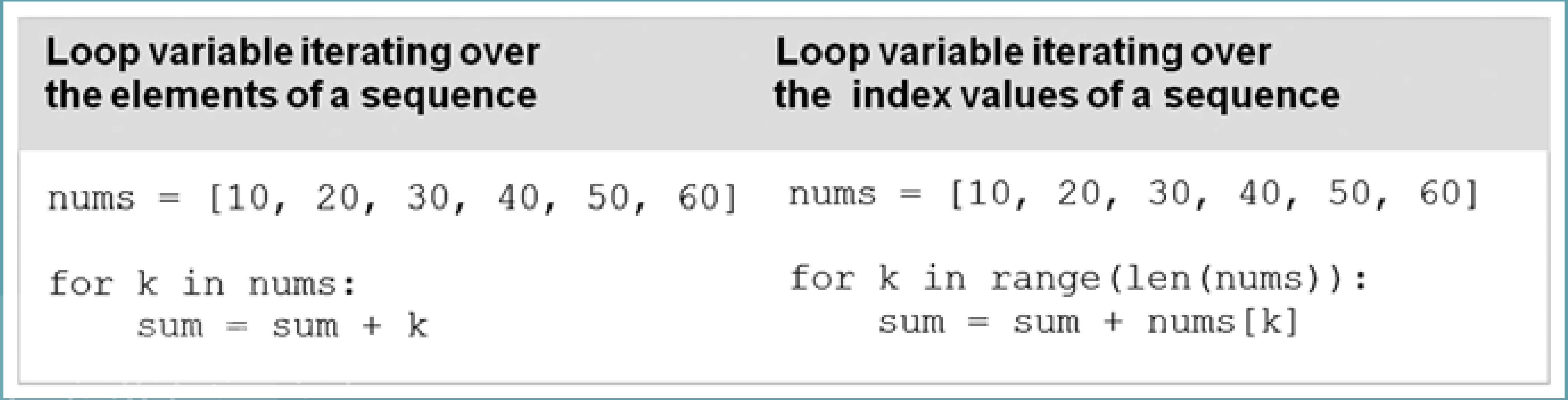
Built-in range Function

- Python provides a built-in range function that can be used for generating a sequence of integers that a for loop can iterate over, as shown below.

```
sum = 0
for k in range(1, 11):
    sum = sum + k
```

Iterating Over List Elements vs. List Index Values

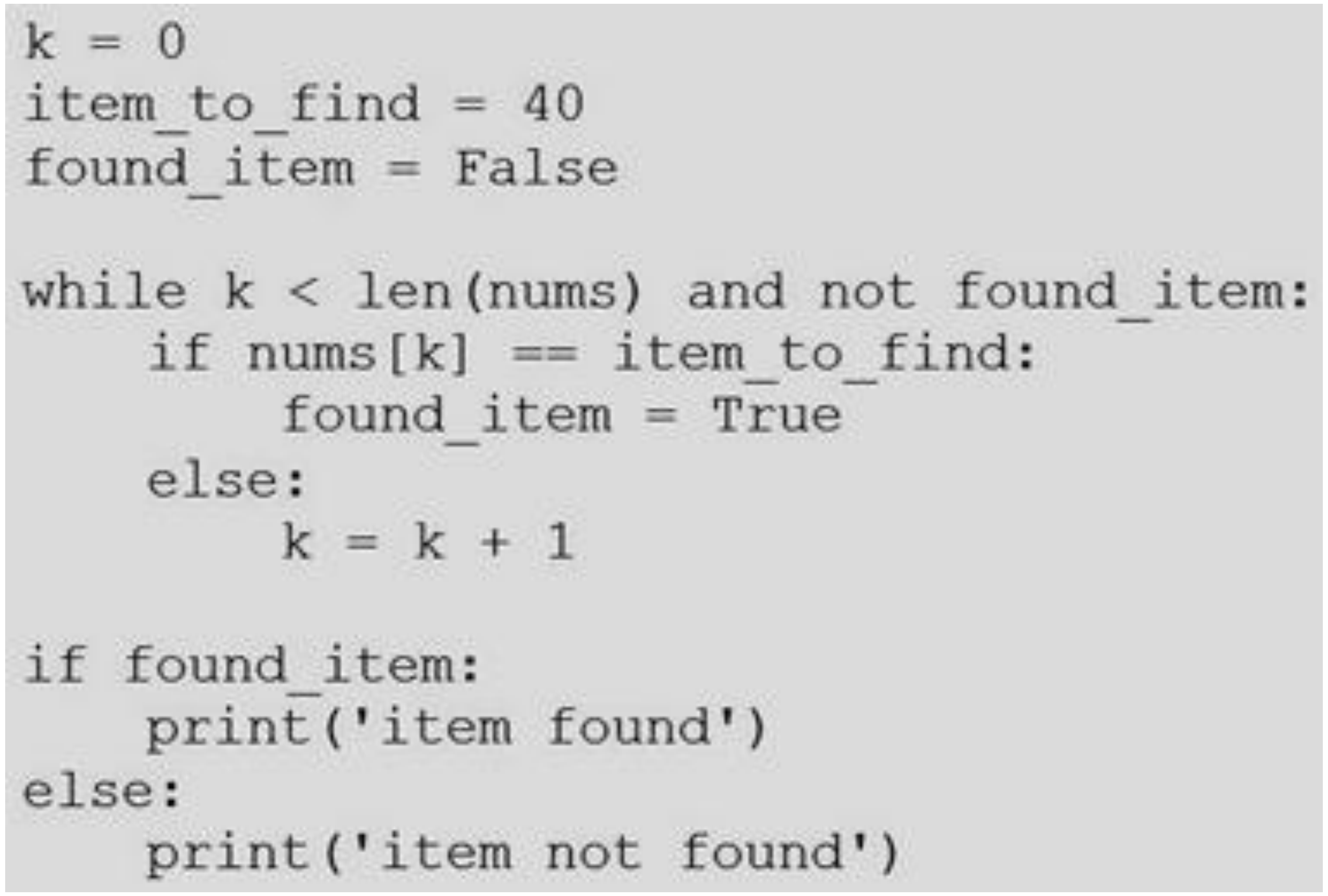
- When the elements of a list need to be accessed, but not altered, a loop variable that iterates over each list element is an appropriate approach.
- However, there are times when the loop variable must iterate over the *index values* of a list instead.



Loop variable iterating over the elements of a sequence	Loop variable iterating over the index values of a sequence
`nums = [10, 20, 30, 40, 50, 60]`	`nums = [10, 20, 30, 40, 50, 60]`

`for k in nums:`
`    sum = sum + k`
	`for k in range(len(nums)):`
	`    sum = sum + nums[k]`

While Loops and Lists



```
k = 0
item_to_find = 40
found_item = False

while k < len(nums) and not found_item:
    if nums[k] == item_to_find:
        found_item = True
    else:
        k = k + 1

if found_item:
    print('item found')
else:
    print('item not found')
```

- There are situations in which a sequence is to be traversed while a given condition is true.
- In such cases, a while loop is the appropriate control structure.

MCQs

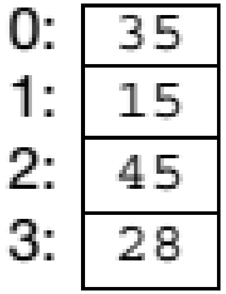
1. What would be the range of index values for a list of 10 elements?

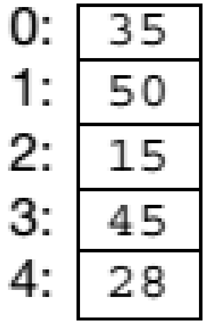
   (a) 0–9 (b) 0–10 (c) 1–10

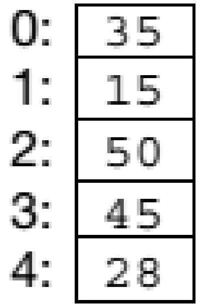
2. Which one of the following is NOT a common operation on lists?

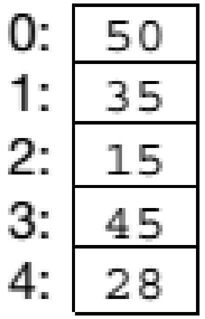
   (a) access (b) replace (c) interleave (d) append (e) insert (f) delete

3. Which of the following would be the resulting list after inserting the value 50 at index 2?



(a)


(b)


(c)


MCQs: Answers

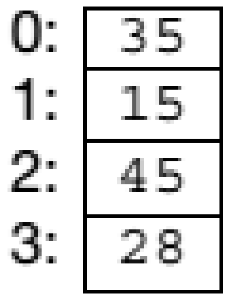
1. What would be the range of index values for a list of 10 elements?

   (a) 0–9 (b) 0–10 (c) 1–10

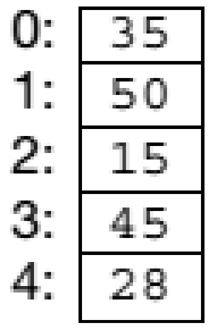
2. Which one of the following is NOT a common operation on lists?

   (a) access (b) replace **(c) interleave** (d) append (e) insert (f) delete

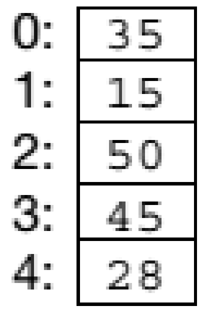
3. Which of the following would be the resulting list after inserting the value 50 at index 2?



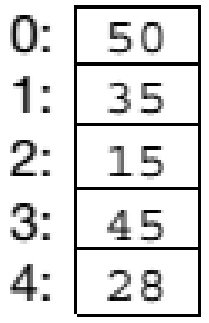
(a)



(b)



(c)


0: 50
1: 35
2: 15
3: 45
4: 28