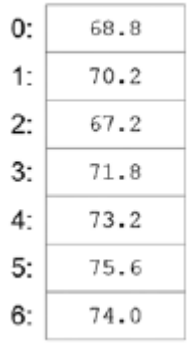
List Structures

In this section we introduce the use of lists in programming. The concept of a list is similar to our everyday notion of a list. We read off (access) items on our to-do list, add items, cross off (delete) items, and so forth.

What Is a List?

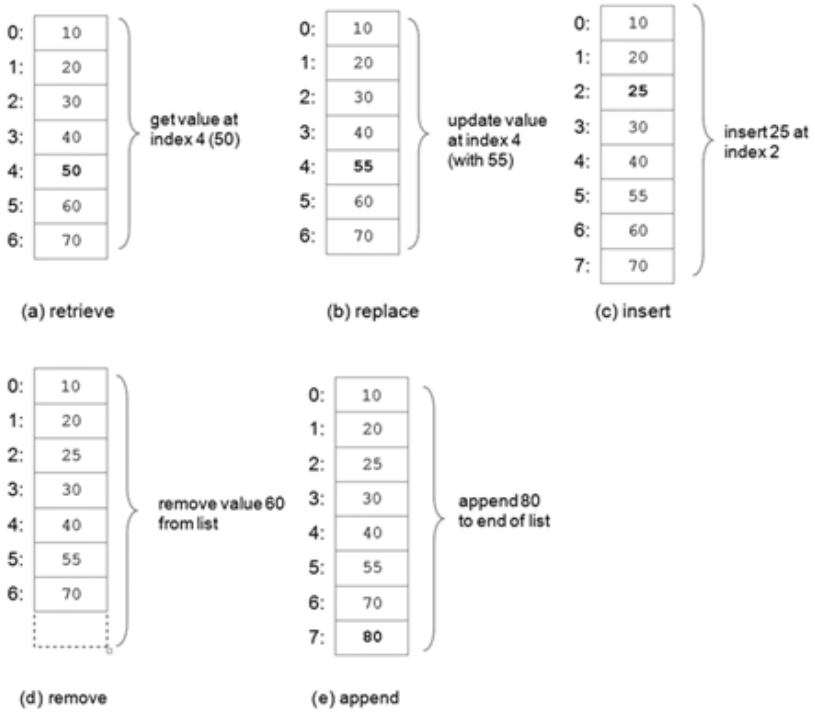
A **list** is a *linear data structure*, meaning that its elements have a linear ordering. That is, there is a first element, a second element, and so on. Figure 4-2 depicts a list storing the average temperature for each day of a given week, in which each item in the list is identified by its *index value*.



The location at index 0 stores the temperature for Sunday, the location at index 1 stores the temperature for Monday, and so on. It is customary in programming languages to begin numbering sequences of items with an index value of 0 rather than 1. This is referred to as *zero-based indexing*. This is important to keep in mind to avoid any “off by one” errors in programs, as we shall see.

Common List Operations

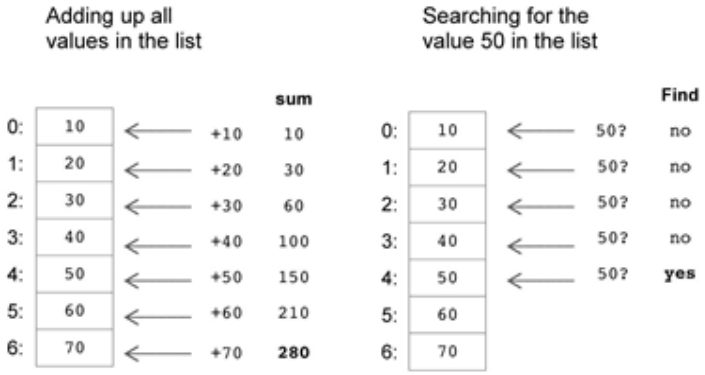
Operations commonly performed on lists include retrieve, update, insert, delete (remove) and append. Figure 4-3 depicts these operations on a list of integers.



The operation depicted in (a) retrieves elements of a list by index value. Thus, the value 50 is retrieved at index 4 (the fifth item in the list). The replace operation in (b) updates the current value at index 4, 50, with 55. The insert operation in (c) inserts the new value 25 at index 2, thus shifting down all elements below that point and lengthening the list by one. In (d), the remove operation deletes the element at index 6, thus shifting up all elements below that point and shortening the list by one. Finally, the append operation in (e) adds a new value, 80, to the end of the list.

List Traversal

A **list traversal** is a means of accessing, one-by-one, the elements of a list. For example, to add up all the elements in a list of integers, each element can be accessed one-by-one, starting with the first, and ending with the last element. Similarly, the list could be traversed starting with the last element and ending with the first. To find a particular value in a list also requires traversal. Depicted below are the tasks of summing and searching a list.



Part II - Lists (Sequences) in Python

A **list** in Python is a mutable, linear data structure of variable length, allowing mixed-type elements. *Mutable* means that the contents of the list may be altered. Lists in Python use zero-based indexing. Thus, all lists have index values `0 ... n-1`, where `n` is the number of elements in the list. Lists are denoted by a comma-separated list of elements within square brackets as shown below,

```
[1, 2, 3]   ['one', 'two', 'three']   ['apples', 50, True]
```

An **empty list** is denoted by an empty pair of square brackets, `[]`. (We shall later see the usefulness of the empty list.) Elements of a list are accessed by using an index value within square brackets,

`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

Thus, for example, the following prints the first element of list `lst`,

```
print(lst[0])
```

The elements in list `lst` can be summed as follows,

```
sum = lst[0] + lst[1] + lst[2]
```

For longer lists, we would want to have a more concise way of traversing the elements. We discuss this below. Elements of a list can be updated (replaced) or deleted (removed) as follows (for `lst = [1, 2, 3]`),

`lst[2] = 4`	`[1, 2, 4]`	replacement of 3 with 4 at index 2
`del lst[2]`	`[1, 2]`	removal of 4 at index 2

Methods `insert` and `append` also provide a means of altering a list,

`lst.insert(1, 3)`	`[1, 3, 2]`	insertion of 3 at index 1
`lst.append(4)`	`[1, 3, 2, 4]`	appending of 4 to end of list

In addition, methods `sort` and `reverse` reorder the elements of a given list. These list modifying operations are summarized BELOW:

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']`

Your Turn

From the Python Shell, enter the following and observe the results.

```
>>> lst = [10, 20, 30]
>>> lst
???

>>> lst [0]
???

>>> lst[0] = 5
>>> lst
???

>>> del lst [2]
>>> lst
???

>>> lst.insert (1 , 15)
>>> lst
???

>>> lst.append (40)
>>> lst
???
```

Part III - Tuples

A **tuple** is an *immutable* linear data structure. Thus, in contrast to lists, once a tuple is defined, it cannot be altered. Otherwise, tuples and lists are essentially the same. To distinguish tuples from lists, tuples are denoted by parentheses instead of square brackets as shown below,

```
nums = (10, 20, 30)
student = ('John Smith', 48, 'Computer Science', 3.42)
```

Another difference between tuples and lists is that *tuples of one element must include a comma following the element.* Otherwise, the parenthesized element will not be made into a tuple, as shown below,

CORRECT	WRONG
`>>> (1,)`	`>>> (1)`
`(1)`	`1`

An empty tuple is represented by a set of empty parentheses, (). The elements of tuples are accessed the same as lists, with square brackets,

```
>>> nums[0]          >>> student[0]
10                   'John Smith'
```

Any attempt to alter a tuple is invalid. Thus, delete, update, insert, and append operations are not defined on tuples. For now, we can consider using tuples when the information to represent should not be altered.

Your Turn

From the Python Shell, enter the following and observe the results.

```
>>> t = (10, 20, 30)          >>> t.insert (1 , 15)
>>> t[0]                      >>> ???
???                           ???

>>> del t[2]                 >>> t.append (40)
???                           ???
```

Part IV - Sequences

A **sequence** in Python is a linearly ordered set of elements accessed by an index number. Lists, tuples, and strings are all sequences. Strings, like tuples, are immutable; therefore, they cannot be altered. Sequence operations common to strings, lists, and tuples in Python are listed below:

Operation		String s = 'hello' w = '!'	Tuple s = (1,2,3,4) w = (5,6)	List s = [1,2,3,4] w = [5,6]
Length	`len(s)`	`5`	`4`	`4`
Select	`s[0]`	`'h'`	`1`	`1`
Slice	`s[1:4]` `s[1:]`	`'ell'` `'ello'`	`(2, 3, 4)` `(2, 3 ,4)`	`[2 ,3, 4]` `[2, 3, 4]`
Count	`s.count('e')` `s.count(4)`	`1` *error*	`0` `1`	`0` `1`
Index	`s.index('e')` `s.index(3)`	`1` `--`	`--` `2`	`--` `2`
Membership	`'h' in s`	`True`	`False`	`False`
Concatenation	`s + w`	`'hello!'`	`(1, 2, 3, 4, 5, 6)`	`[1, 2, 3, 4, 5, 6]`
Minimum Value	`min(s)`	`'e'`	`1`	`1`
Maximum Value	`max(s)`	`'o'`	`4`	`4`
Sum	`sum(s)`	*error*	`10`	`10`

For any sequence `s`, `len(s)` gives its length, and `s[k]` retrieves the element at index `k`. The slice operation, `s[index1:index2]`, returns a subsequence of a sequence, starting with the first index location up to *but not including* the second. The `s[index:]` form of the slice operation returns a string containing all the list elements starting from the given index location to the end of the sequence. The `count` method returns how many instances of a given value occur within a sequence, and the `find` method returns the index location of the *first occurrence* of a specific item, returning 21 if not found. For determining only if a given value occurs within a sequence, without needing to know where, the `in` operator (introduced in Chapter 3) can be used instead.

The + operator is used to denote concatenation. Since the plus sign also denotes addition, Python determines which operation to perform based on the operand types. Thus the plus sign, +, is referred to as an overloaded operator. If both operands are numeric types, addition is performed. If both operands are sequence types, concatenation is performed. (If a mix of

numeric and sequence operands is used, an "`unsupported operand type(s) for +`" error message will occur.) Operations `min`/`max` return the smallest/largest value of a sequence, and `sum` returns the sum of all the elements (when of numeric type). Finally, the comparison operator, 55, returns `True` if the two sequences are the same length, and their corresponding elements are equal to each other.

Your Turn

From the Python Shell, enter the following and observe the results.

```
>>> s = 'coconut'              >>> s = [10, 20, 30, 10]
>>> s [4:7]                    >>> s [1:3]
???                             ???

>>> s.count ('o')             >>> s.count (10)
???                             ???

>>> s.index ('o')             >>> s.index (10)
???                             ???

>>> s + 'juice'               >>> s + (40,50)
???                             ???
```

Part V - Nested Lists

Lists and tuples can contain elements of any type, including other sequences. Thus, lists and tuples can be nested to create arbitrarily complex data structures. Below is a list of exam grades for each student in a given class,

```
class_grades 5 [ [85, 91, 89], [78, 81, 86], [62, 75, 77],...]
```

In this list, for example, `class_grades[0]` equals `[85, 91, 89]`, and `class_grades[1]` equals `[78, 81, 86]`. Thus, the following would access the first exam grade of the first student in the list,

```
student1_grades = class_grades[0]
student1_exam1 = student1_grades[0]
```

However, there is no need for intermediate variables `student1_grades` and `student1_exam1`. The exam grade can be directly accessed as follows,

class_grades[0][0] → [85, 91, 89][0] → 85

To calculate the class average on the first exam, a while loop can be constructed that iterates over the first grade of each student's list of grades,

```
sum = 0
k = 0
while k < len(class_grades):
    sum = sum + class_grades[k][0]
    k =  k + 1
average_exam1 =  sum / float(len(class_grades))
```

If we wanted to produce a new list containing the exam average for each student in the class, we could do the following,

```
exam_avgs = []
k = 0
while k < len(class_grades):
    avg = (class_grades[k][0] + class_grades[k][1] 1 \
              class_grades[k][2]) / 3.0
    exam_avgs.append(avg)
    k =  k + 1
```

Each time through the loop, the average of the exam grades for a student is computed and appended to list `exam_avgs`. When the loop terminates, `exam_avgs` will contain the corresponding exam average for each student in the class.

Your Turn

From the Python Shell, enter the following and observe the results.

```
>>> lst = [1, 2 , 3], [4, 5, 6], [7, 8, 9]
>>> lst[0]                    >>> lst [1]
???                           ???

>>> lst [0][1]               >>> lst [1] [1]
???                          ???
```

Concepts and Procedures

1. Which of the following sequence types is a mutable type?
(a) strings (b) lists (c) tuples

2. Which of the following is true?
a) Lists and tuples are denoted by the use of square brackets.
b) Lists are denoted by use of square brackets and tuples are denoted by the use of parentheses.
c) Lists are denoted by use of parentheses and tuples are denoted by the use of square brackets.

3. Lists and tuples must each contain at least one element. (TRUE/FALSE)

4. For `lst = [4, 2, 9, 1]`, what is the result of the following operation, `lst.insert(2, 3)`?

(a) `[4, 2, 3, 9, 1]` **(b)** `[4, 3 ,2, 9, 1]` **(c)** `[4, 2, 9, 2, 1]`

5. Which of the following is the correct way to denote a tuple of one element?
(a) [6] (b) (6) (c) [6,] (d) (6,)

6. Which of the following set of operations can be applied to any sequence?
a) `len(s), s[i], s + w (concatenation)`
b) `max(s), s[i], sum(s)`
c) `len(s), s[i], s.sort()`

7. What would be the range of index values for a list of 10 elements?
(a) 0–9 (b) 0–10 (c) 1–10

8. Which one of the following is NOT a common operation on lists?
(a) access (b) replace (c) interleave
(d) append (e) insert (f) delete

9. Which of the following lists are syntactically correct in Python?
(a) `[1, 2, 3, 'four']` (b) `[1, 2, [3, 4]]` (c) `[[1, 2, 3]['four']]`

10. For `lst = [4, 2, 9, 1]`, what is the result of each of the following list operations?
(a) `lst[1]` (b) `lst.insert(2, 3)` (c) `del lst[3]` (d) `lst.append(3)`

11. For `fruit = ['apple', 'banana', 'pear', 'cherry']`, use a list operation to change the list to `['apple', 'banana', 'cherry']`.

12. For a list of integers, `lst`, give the code to retrieve the maximum value of the second half of the list.

13. For variable `product_code` containing a string of letters and digits,

a) Give an if statement that outputs `"Verified"` if `product_code` contains both a "z" and a "9", and outputs `"Failed"` otherwise.

b) Give a Python instruction that prints out just the last three characters in `product_code`.

14. Which of the following are valid operations on tuples (for tuples `t1` and `t2`)?

(a) `len(t1)` (b) `t1 + t2` (c) `t1.append(10)` (d) `t1.insert(0, 10)`

15. For `str1 = 'Hello World'`, answer the following,

a) Give an instruction that prints the fourth character of the string.

b) Give an instruction that finds the index location of the first occurrence of the letter 'o' in the string.

16. For a nested list `lst` that contains sublists of integers of the form `[n1, n2, n3]`,

a) Give a Python instruction that determines the length of the list.

b) Give Python code that determines how many total integer values there are in list `lst`.

c) Give Python code that totals all the values in list `lst`.

d) Given an assignment statement that assigns the third integer of the fourth element (sublist) of `lst` to the value 12.

Problem Solving

1. Write a Python program that prompts the user for a list of integers, stores in another list only those values that are in tuple `valid_values`, and displays the resulting list.