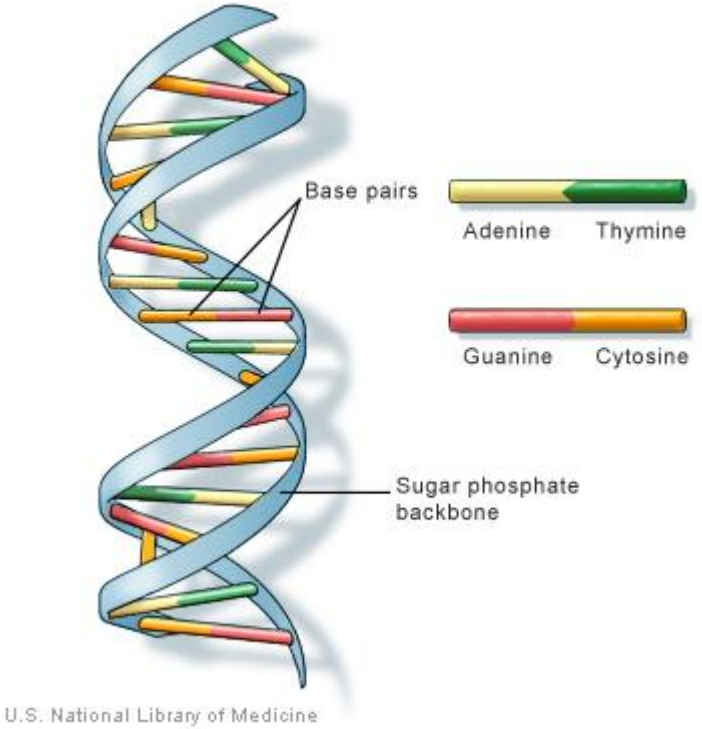
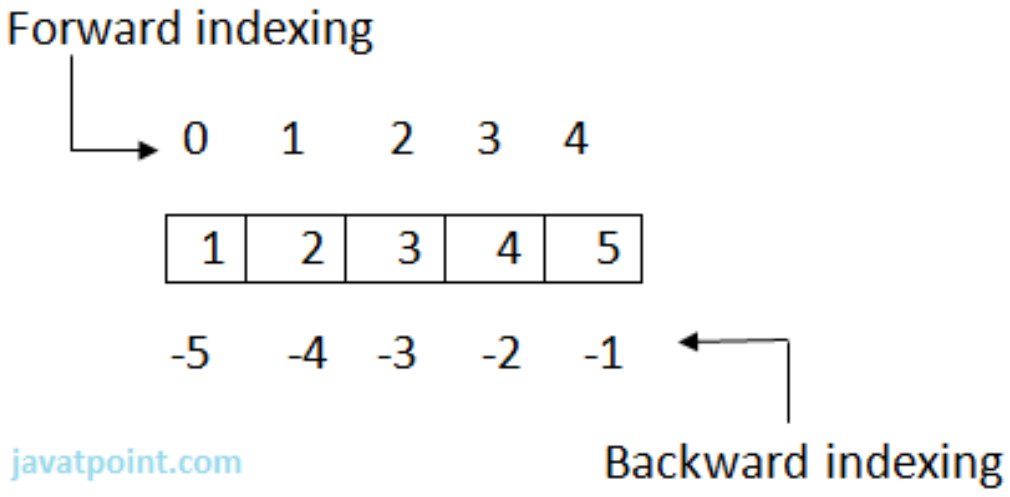
List

List everywhere

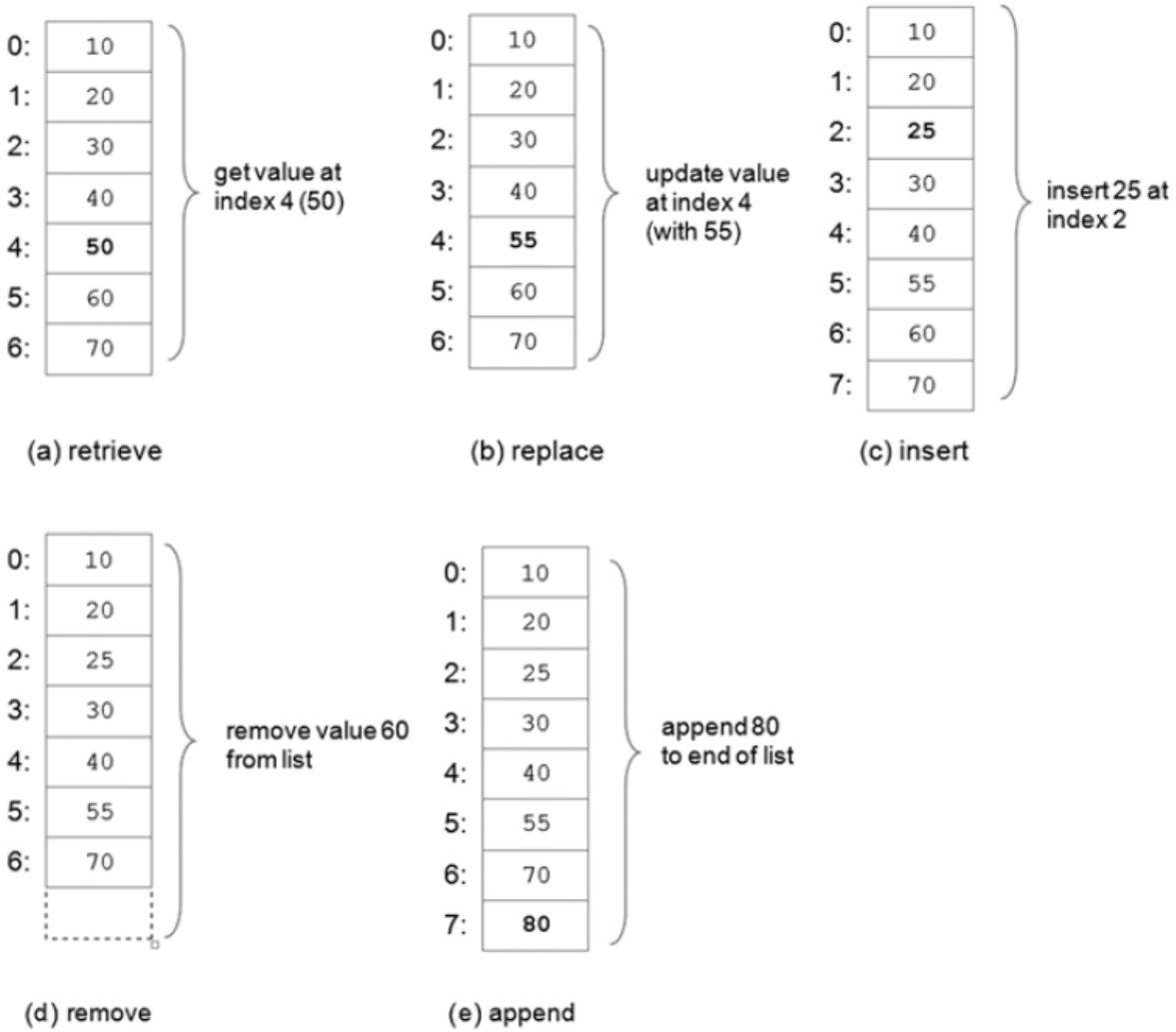


List structure

- A **list** is a linear data structure, thus its elements have a linear ordering.



Operations on lists



List transversal

A **list traversal** is a means of accessing, one-by-one, the elements of a list.

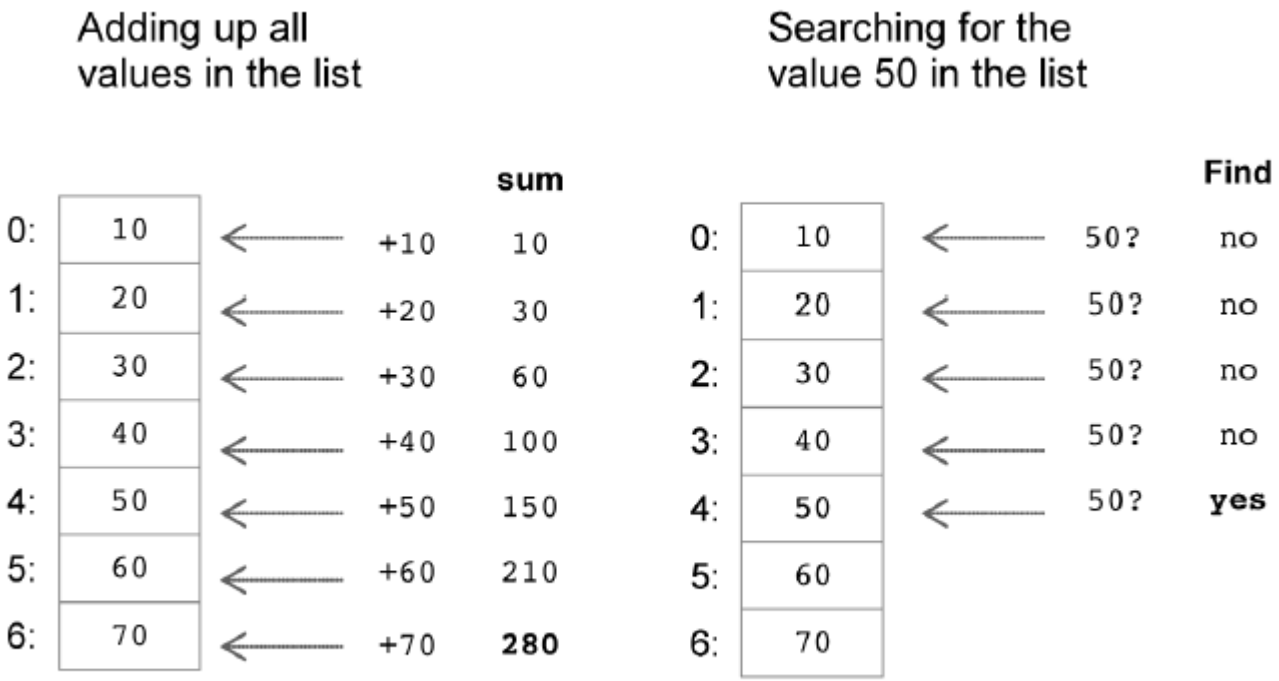


FIGURE 4-4 List Traversal

List properties

List Characteristics	Elements
Element Type	All elements of the same type
	Elements of different types
Length	Fixed length
	Varying length
Modifiability	Mutable (alterable)
	Immutable (unalterable)
Common Operations	Determine if a list is empty
	Determine the length of a list
	Access (retrieve) elements of a list
	Insert elements into a list
	Replace elements of a list
	Delete elements of a list
	Append elements to (the end of) a list

Lists 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 in Python

```
[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
```

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

```
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
```

```
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
```

Lists 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']`

LET'S TRY IT

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
???
```

Tuples in Python

- A **tuple** is an *immutable* linear data structure

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

- *Tuples of one element must include a comma following the element*

CORRECT

```
>>> (1,)
(1)
```

WRONG

```
>>> (1)
1
```

LET'S TRY IT

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

```
>>> t = (10, 20, 30)
>>> t[0]
???

>>> del t[2]
???

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

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

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

Sequences operations

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`

Sequence operations

LET'S TRY IT

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

```
>>> s = 'coconut'
>>> s[4:7]
???

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

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

>>> s + ' juice'
???
```

```
>>> s = (10, 30, 20, 10)
>>> s[1:3]
???

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

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

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

```
>>> s = [10, 30, 20, 10]
>>> s[1:3]
???

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

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

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

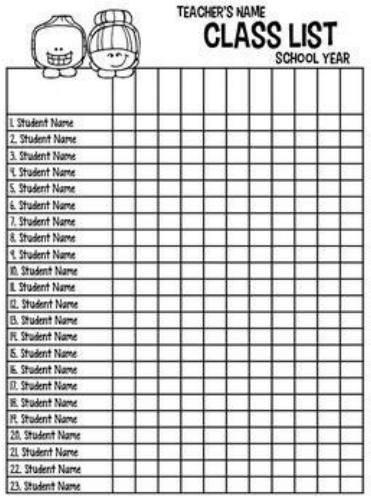
Nested lists

- Lists and tuples can be nested within each other to construct arbitrarily complex data structures.

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

student1_grades = class_grades[0]

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



Nested lists

```
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))
```

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

Iterating over lists

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

for statement	Example use

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

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

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

LET'S TRY IT

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

```
>>> for k in [4, 2 ,3, 1]:
        print(k)
???

>>> for k in (4, 2, 3, 1):
        print(k)
???
```

```
>>> for k in ['Apple', 'Banana', 'Pear']:
        print(k)
???

>>> for k in 'Apple':
        print(k)
???
```

Range

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

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

LET'S TRY IT

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

```
>>> for k in range(0, 11):
        print(k)
???

>>> for k in range[0, 11]:
        print(k)
???

>>> for k in range(2, 102, 2):
        print(k)
???

>>> for k in range(10, -1, -2):
        print(k)
???
```

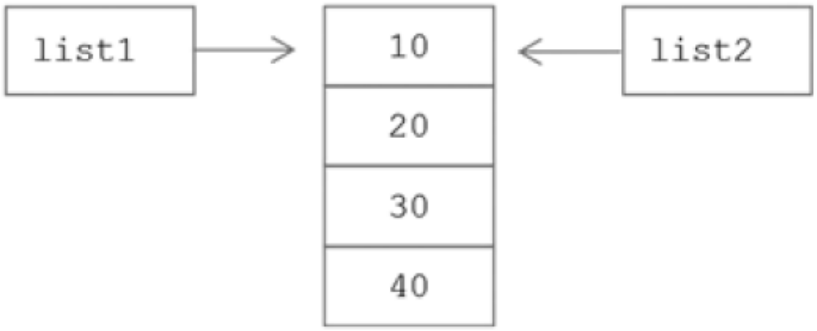
Iteration over list vs list element values

An **index variable** is a variable whose changing value is used to access elements of an indexed data structure.

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]`
 	
`for k in nums:`
`    sum = sum + k`
	`nums = [10, 20, 30, 40, 50, 60]`
	
	`for k in range(len(nums)):`
	`    sum = sum + nums[k]`

Copying lists

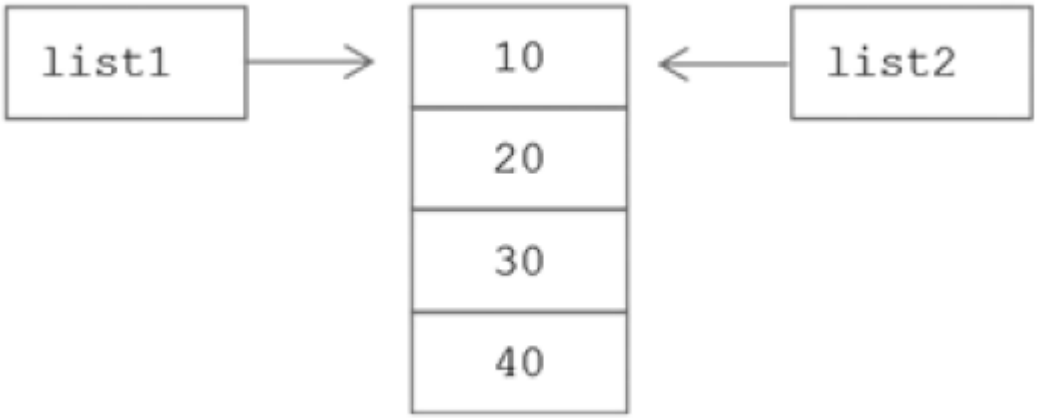



```
>>> list1 = [10, 20, 30, 40]
>>> list2 = list1
>>> list1[0] = 5
>>> list1
[5, 20, 30, 40]
>>> list2
[5, 20, 30, 40]
```

change made in list1

change in list1 causes a change in list2

Copying lists




```
>>> list1 = [10, 20, 30, 40]
>>> list2 = list(list1)
>>> list1[0] = 5
>>> list1
[5, 20, 30, 40]
>>> list2
[10, 20, 30, 40]
```

change made in list1

change in list1 does NOT cause any change in list2

List comprehentions

- **List comprehensions** in Python provide a concise means of generating a more varied set of sequences than those that can be generated by the range function.

Example List Comprehensions	Resulting List
(a) `[x**2 for x in [1, 2, 3]]`	`[1, 4, 9]`
(b) `[x**2 for x in range(5)]`	`[0, 1, 4, 9, 16]`
(c) `nums = [-1, 1, -2, 2, -3, 3, -4, 4]` `[x for x in nums if x >= 0]`	`[1, 2, 3, 4]`
(d) `[ord(ch) for ch in 'Hello']`	`[72, 101, 108, 108, 111]`

(e) `vowels = ('a', 'e', 'i', 'o', 'u')`
`w = 'Hello'`
`[ch for ch in w if ch in vowels]`
	`['e', 'o']`