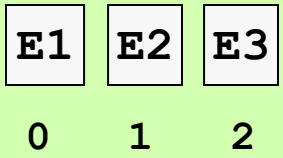
IST 1025

Introduction to Programming

Sequences: Lists

What Is a List?

- A *list* is a sequence of 0 or more data values (of any types) called *elements*



- The programmer can access or replace the element at any position in a list

- An element can be inserted at any position or removed from any position

Real-World Examples

- A shopping list
- A schedule of athletic contests
- A team roster
- An algorithm (a list of instructions)

Literals, Assignment, Comparisons, Concatenation, `for` Loop

```
a = [1, 2, 3]

b = list(range(1, 4))  # b now refers to [1, 2, 3]

a == b                 # returns True
a < [2, 3, 4]          # returns True

print(a + b)           # displays [1, 2, 3, 1, 2, 3]

print(len(a))          # displays 3

for item in [1, 2, 3]: print(item)
```

Similar to the behavior of strings so far

Indexing and Slicing

```
a = [1, 2, 3]

print(a[2])                 # Displays 3

print(a[len(a) - 1])        # Displays 3

print(a[-1])                # Displays 3

print(a[0:2])               # Displays [1, 2]
```

Similar to the behavior of strings so far

Three Ways to Get a Sum

```
a = [1, 2, 3]

total = 0
for index in range(len(a)):
    total += a[index]
```

Three Ways to Get a Sum

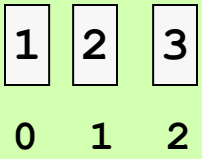
```
a = [1, 2, 3]

total = 0
for index in range(len(a)):
    total += a[index]


total = 0
for item in a:
    total += item
```

Three Ways to Get a Sum

```
a = [1, 2, 3]

total = 0
for index in range(len(a)):
    total += a[index]


total = 0
for item in a:
    total += item


total = sum(a)
```

Replacing an Item

`[1, 2, 3]`



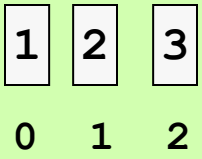
To replace an item at a given position, use the subscript operator with the appropriate index

```
<a list>[<an int>] = <an expression>

a = [1, 2, 3]

a[1] = 5          # The list is now [1, 5, 3]

print(a[1])        # Displays 5
```

Unlike strings and tuples, lists are mutable!

Increment ‘Em All

`[1, 2, 3]`



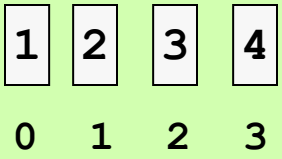

```
a = [1, 2, 3]

for index in range(len(a)):
    a[index] = a[index] + 1
```

Cannot use an item-based **`for`** loop for replacements

Must use an index-based loop

Replacing a Subsequence

```
[1, 2, 3, 4]
```




```
a = [1, 2, 3, 4]

a[0:2] = [5, 6]

print(a)                    # Displays [5, 6, 3, 4]
```

Splitting

`split` builds a list of tokens (words) from a string using the space or newline as the default separator

```
s = 'Python is way cool!'

lyst = s.split()

print(lyst)                    # Displays ['Python', 'is', 'way', 'cool!']
```

***<a string>*.split(<*optional separator string*>)**

Pattern Matching

```
lyst = ['Ken', 100]

[name, grade] = lyst

print(name)             # Displays Ken

print(grade)            # Displays 100
```

Application: Find the Highest Grade

```
fileName = input('Enter the file name: ')
inputFile = open(fileName, 'r')
highestGrade = 0
topStudent = 'Nobody'
for line in inputFile:
    [name, grade] = line.split()
    grade = int(grade)
    if grade > highestGrade:
        highestGrade = grade
        topStudent = name
print(topStudent, 'has the highest grade', highestGrade)
```

Assumes that each line of text in the file contains two words, a name and a grade (represented as an integer)

Joining

`join` builds a string from a list of tokens (words)

```
s = 'Python is way cool!'

lyst = s.split()

print(lyst)              # Displays ['Python', 'is', 'way', 'cool!']

print(' '.join(lyst))    # Displays Python is way cool!
```

***<a separator string>*.join(<*a list of strings*>)**

Application: Sentence Length

Short sentences are an index of good writing style. Word processing programs allow you to do word counts.

```
sentence = input('Enter a sentence: ')

words = sentence.split()

count = len(words)

print('There are', count, 'words in your sentence.')
```

Just the Tip of the Iceberg

- The list is a very powerful data structure
- There are many list processing methods
- A Python *method* is like a function, but uses a slightly different syntax

The **append** Method

```
lyst = [1, 2, 3]

lyst.append(4)

print(lyst)                  # Displays [1, 2, 3, 4]
```

Adds an element to the end of the list

Syntax for calling the **append** method:

```
<a list>.append(<an element>)   # Puts the element at the end of the list
                                # Actually modifies the list!!!
```

Functions vs Methods

```
lyst = [1, 2, 3]

lyst.append(4)                          # A method call

print(len(lyst))                        # Two function calls

file = open('testfile.txt', 'r')        # A function call

wordList = file.read().split()          # Two method calls
```

Syntax of method calls and function calls:

```
<a data object>.<method name>(<arguments>)

<function name>(<arguments>)
```

Some List Methods

Example Call	What It Does
`lyst.count(3)`	Returns the number of 3s in the list
`lyst.insert('dog', 2)`	Inserts `'dog'` at position 2, after shifting the elements at positions 2 through N - 1 to the right
`lyst.pop(0)`	Removes the element at the first position and then shifts the remaining elements to the left by one position
`lyst.remove('dog')`	Removes the first instance of `'dog'` in the list
`lyst.reverse()`	Reverses the elements
`lyst.sort()`	Sorts the elements in ascending order