Lists

Arrays and Lists

• From the earliest days of computing, programming languages have supported the idea of an array, which is an ordered sequence of values.
• The individual values in an array are called elements. The number of elements is called the length of the array.
• Each element is identified by its position number in the array, which is called its index. In Python—as in almost all modern languages—index numbers begin with 0 and extend up to one less than the length of the array.
• Python implements the array concept in a more general form called a list. Lists support all standard array operations, but also allow insertion and deletion of elements.

Creating Lists in Python

• The simplest way to create a list in Python is to specify its elements surrounded by square brackets and separated by commas. For example, the declaration

```
COIN_VALUES = [ 1, 9, 10, 25, 50, 100 ]
```

creates a constant list of six elements that correspond to the standard coins available in the United States.
• Lists are most commonly represented conceptually as a series of numbered boxes, as in the following representation of COIN_VALUES:

<table>
<thead>
<tr>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1</td>
<td>5</td>
<td>10</td>
<td>25</td>
<td>50</td>
</tr>
</tbody>
</table>

Nonnumeric Lists

• Lists may contain values of any Python type. For example, the declaration

```
COIN_NAMES = [ "penny", "nickel", "dime", "quarter", "half-dollar", "dollar" ]
```

creates the following list:

<table>
<thead>
<tr>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>penny</td>
<td>nickel</td>
<td>dime</td>
<td>quarter</td>
<td>half-dollar</td>
</tr>
</tbody>
</table>

List Selection

• Given an array, you can get the value of any element by writing the index of that element in brackets after the array name. This operation is called selection.
• For example, given the declarations on the preceding slides, the value of COIN_VALUES[3] is 25.
• Similarly, the value of COIN_NAMES[2] is the string "dime".

```
COIN_VALUES
<table>
<thead>
<tr>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1</td>
<td>5</td>
<td>10</td>
<td>25</td>
<td>50</td>
</tr>
</tbody>
</table>

COIN_NAMES
<table>
<thead>
<tr>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>penny</td>
<td>nickel</td>
<td>dime</td>
<td>quarter</td>
<td>half-dollar</td>
</tr>
</tbody>
</table>
```

Graphics Contest

The CSCI 121 Graphics Contest

Fall 2019 Results
Cycling through List Elements

- One of the most useful idioms for a list is iterating through each of its elements in turn. The standard `for` loop pattern for doing so looks like this:

```python
for variable in list:
    Perform some operation on this variable.
```

- As an example, the following function returns the sum of the elements in the list:

```python
def sumIntegerList(list):
    sum = 0
    for value in list:
        sum += value
    return sum
```

Sequences

- The last few slides should remind you of the operations for strings, which are almost exactly the same.
- Strings and lists are both examples of a more general class of objects in Python called `sequences`. All sequences support the following operations:
  - The `len` function
  - Index numbering beginning at 0
  - Negative index numbering that counts backward from the end
  - Selection of an individual element using square brackets
  - Slicing in all its forms
  - Concatenation using the `+` or `+=` operator
  - Repetition using the `*` operator
  - Inclusion testing using the `in` operator

Mutable vs. Immutable Types

- The most important difference between a list and a string is that you are allowed to change the contents of a list while the characters in a string are fixed.
- Types like strings for which you are not allowed to change the individual components are defined to be `immutable`.
- Types like lists where the elements are assignable are said to be `mutable`.
- Immutable types have many advantages in programming:
  - You don’t have to worry about whether values will be changed.
  - Values that are immutable can more easily be shared.
  - Immutable objects are easier to use in concurrent programs.
- Despite these advantages, there are still situations in which mutable types like lists are just the right tools.

Passing Lists as Parameters

- When you pass a list as a parameter to a function or return a list as a result, only the reference to the list is actually passed between the functions.
- The effect of this strategy of representing lists as references is that the elements of a list are effectively shared between a function and its caller. If a function changes an element of a list passed as a parameter, that change will persist after the function returns.
- The next slide simulates a program that does the following:
  1. Initializes a list to contain the integers from 0 to N-1.
  2. Prints the elements in the list.
  3. Reverses the elements in the list.
  4. Prints the reversed list on the console.

Trace of the `reverseList` Function

```python
def reverseList(list):
    for lb in range(len(list) // 2):
        rb = len(list) - lb - 1
        list[lb], list[rb] = list[rb], list[lb]
```

Methods that Return Information

- `list.index(value)`
  - Returns the first index at which `value` appears in `list` or raises an error.
- `list.index(value, start)`
  - Returns the first index of `value` after the starting position.
- `list.count(value)`
  - Returns the number of times `value` appears in `list`.
- `list.copy()`
  - Creates a new list whose elements are the same as the original.
Methods that Add and Remove Elements

- `list.append(value)` Adds `value` to the end of the list.
- `list.insert(index, value)` Inserts `value` before the specified index position.
- `list.remove(value)` Removes the first instance of `value`, or raises an error if it’s not there.
- `list.pop()` Removes and returns the last element of the list.
- `list.pop(index)` Removes and returns the element at the specified index.
- `list.clear()` Removes all elements from the list.

Methods that Reorder Elements

- `list.reverse()` Reverses the order of elements in the list.
- `list.sort()` Sorts the elements of list in increasing order.
- `list.sort(key)` Sorts the elements of list using `key` to generate the key value.
- `list.sort(key, reverse)` Sorts in descending order if `reverse` is `True`.

Functions on Iterators

- The `reverse` and `sort` methods from the preceding slide do not return a value but instead change the elements of the list to which those methods are applied.
- What you want in many applications is to iterate through the elements of a list in a particular order without changing the contents of the list.
- Python provides two functions that take an iterable object as a parameter and return another iterable object that processes the elements in a different order:
  - The `reversed` function creates an iterable object that returns its elements in the opposite order.
  - The `sorted` function creates an iterable object that returns its elements in ascending order.
- Each of these functions leaves its arguments unchanged.

Exercise: `reversed` and `sorted`

- How would you write a `for` loop that cycled through the elements of an array named `scores` in sorted order?
- The idiomatic loop header
  ```python
  for i in range(len(list)):
  ```
  repeatedly executes the loop body with `i` set to every index position in `list`, starting at 0 and continuing up to the length of the list minus one. How would you write a loop header that cycled through these index positions in reverse order?
- Describe the differences between the built-in `sorted` function and the `sort` method in the `list` class.

List Methods that Involve Strings

- `str.split()` Splits a string into a list of its components using whitespace as separator.
- `str.split(sep)` Splits a string into a list using the specified separator.
- `str.splitlines()` Splits a string into separate lines at instances of the newline character.
- `str.join(list)` Joins the elements of list into a string, using `sep` as the separator.