Lecture 8 — Lists Part 1

Overview

- So far we've looked at working with individual values and variables.
- This is cumbersome even for just two or three variables.
- We need a way to aggregate multiple values and refer to them using a single variable.
- We have done a little bit of this with strings and tuples, but now we are going to get started for real.

This lecture is largely based on Sections 8.1-8.3 of *Practical Programming*.

Lists are Sequences of Values

- Gather together values that have common meaning.
- As a first example, here are scores of 7 judges for the free skating part of a figure skating competition:

```
scores = [ 59, 61, 63, 63, 68, 64, 58 ]
```

- As a second example, here are the names of the planets in the solar system (including Pluto, for now):

```
planets = [ 'Mercury', 'Venus', 'Earth', 'Mras', 'Jupiter',
    'Saturn', 'Neptune', 'Uranus', 'Pluto' ]
```

- Notes on syntax:
    - Begin with `[` and end with `]`
    - Commas separate the individual values
    - The spaces between values are optional and are used for clarity here.
    - Any type of object may be stored in a list, and each list can mix different types.

Why bother?

- Gather common values together, providing them with a common name, **especially** when we don't know how many values we will have.
- Apply an operation to the values as a group.
- Apply an operation to each value in the group.
- Examples of computations on lists:
    - Average and standard deviation
    - Which values are above and below the average
    - Correct mistakes
    - Remove values (Pluto)
    - Look at differences
- Watch for these themes throughout the next few lectures.

Accessing Individual Values — Indexing

- Notice that we made the mistake in typing `'Mras'`. How do we fix this? We'll start by looking at *indexing*.
- The line

  ```
  >>> print(planets[1])
  ```

  accesses and prints the string at what's known as index 1 of the list `planets`.
- Each item / value in the list is associated with a unique index
- Indexing in Python (and most other programming languages) starts at 0.
- The notation is again to use `[` and `]` with an integer (non-negative) to indicate which list item.
- What is the last index in `planets`?
    - We can find the length using `len()` and then figure out the answer.

A Memory Model for Lists

We'll draw a memory model in class that illustrates the relationship among

- The name of the list
- The indices
- The values stored in the list

Practice Problems

We will work on these in class:

1. What is the index of the first value in `scores` that is greater than 65?
2. Write a line of Python code to print this value and to print the previous and next values of the list.
3. What is the index of the middle value in a list that is odd length? For even length lists, what are the indices of the middle two values?

Changing Values in the List

- Once we know about indexing, changing a value in a list is easy:

```
>>> planets[3] = 'Mars'
```

- This makes item 3 of `planets` now refer to the string `'Mars'`
- Now we can check the output:

```
>>> print(planets)
```

  to make sure we got it right.
- Strings are similar in many ways.

```
>>> s = 'abc'
>>> s[0]
'a'
>>> s[1]
'b'
```

- Big difference: you can change a part of a list; you cannot change part of a string!

```
>>> s[1] = 'A'
Traceback (most recent call last):
File "<stdin>", line 1, in <module>
TypeError: 'str' object does not support item assignment
```

All Indices Are Not Allowed

- If `t` is a list, then the items are stored at indices from 0 to `len(t)-1`.
- If you try to access indices at `len(t)` or beyond, you get a run-time error. We'll take a look and see.
- If you access negative indices, interesting things happen:

  ```
  >>> print(planets[-1])
  Pluto
  ```

- More specifically, for any list `t`, if `i` is an index from 0 to `len(t)-1` then `t[i]` and `t[i-len(t)]` are the same spot in the list.

Functions on Lists: Computing the Average

- There are many functions (methods) on lists. We can learn all about them using the `help` command.
  - This is just like we did for strings and for modules, e.g.

    ```
    >>> import math
    >>> help(math)

    >>> help(str)
    ```

- Interestingly, we can run help in two ways, one

  ```
  help(list)
  ```

  gives us the list methods, and the second

  ```
  help(planets)
  ```

  tells us that `planets` is a list before giving us list methods.
- First, let's see some basic functions on the list values.
- The basic functions `max`, `sum` and `min` may be applied to lists as well.
- This gives us a simple way to compute the average of our list of scores.

```
>>> print("Average Scores = {:.2f}".format( sum(scores) / len(scores) ))
Average Scores = 62.29
>>> print("Max Score =", max(scores))
Max Score = 68
>>> print("Min Score =", min(scores))
Min Score = 58
```

- Exploring, we will look at what happens when we apply `sum`, `max` and `min` to our list of planet names. Can you explain the result?

Functions that modify the input: Sorting a list

- We can also sort the values in a list by sorting it. Let's try the following:

```
>>> planets = [ 'Mercury', 'Venus', 'Earth', 'Mras', 'Jupiter', \
'Saturn', 'Neptune', 'Uranus', 'Pluto' ]
>>> planets
['Mercury', 'Venus', 'Earth', 'Mras', 'Jupiter', 'Saturn', 'Neptune', 'Uranus',
'Pluto']
>>> planets.sort()
>>> planets
['Earth', 'Jupiter', 'Mercury', 'Mras', 'Neptune', 'Pluto', 'Saturn', 'Uranus',
'Venus']
```

- Note that we did not assign the value returned by sort to a new variable. This is the first function we have learned outside of our Image module that modifies the input but returns nothing. Try the following and see what happens:

```
>>> scores = [ 59, 61, 63, 63, 68, 64, 58 ]
>>> new_scores = scores.sort()
>>> scores
[58, 59, 61, 63, 63, 64, 68]
>>> new_scores
>>>
```

  - Ok, what is the value of the variable `new_scores`? It is unclear. Let's try in a different way.

```
>>> print(scores)
[58, 59, 61, 63, 63, 64, 68]
>>> print(new_scores)
None
>>>
```

- So, the function returns nothing! But, it does change the value of the input list.

- It does so because lists are containers, and functions can manipulate what is inside containers. Functions cannot do so for simple types like integer and float.
- If we want a new list that is sorted without changing the original list then we use the `sorted()` function:

```
>>> scores = [ 59, 61, 63, 63, 68, 64, 58 ]
>>> new_scores = sorted(scores)
>>> scores
[ 59, 61, 63, 63, 68, 64, 58 ]
>>> new_scores
[58, 59, 61, 63, 63, 64, 68]
>>>
```

More Functions: Appending Values, Inserting Values, Deleting

- Now, we will see more functions that can change the value of a list without returning anything.
- Armed with this knowledge, we can figure out how to add and remove values from a list:
  - `append()`
  - `insert()`
  - `pop()`
  - `remove()`
- These operations are fundamental to any “container” — an object type that stores other objects.
  - Lists are our first example of a container

Lists of Lists

- Note that lists can contain any mixture of values, including other lists.
- For example, in

```
>>> L = [ 'Alice', 3.75, ['MATH', 'CSCI', 'PSYC' ], 'PA' ]
```

  - `L[0]` is the name,
  - `L[1]` is the GPA
  - `L[2]` is a list of courses
  - `L[2][0]` is the 0th course, `'MATH'`
  - `L[3]` is a home state abbreviation

- We will write code to print the courses, to change the math course to a stats course, and to append a zipcode.

Additional Practice Problems

1. Write three different ways of removing the last value — `'Pluto'` — from the list of planets. Two of these will use the method `pop`.
2. Write code to insert `'Asteroid belt'` between `'Mars'` and `'Jupiter'`.

Summary

- Lists are sequences of values, allowing these values to be collected and processed together.
- Individual values can be accessed and changed through indexing.
- Functions and methods can be used to **return** important properties of lists like `min()`, `max()`, `sum()`.
- Functions and methods can be also used to **modify** lists, but not return anything.