Topic 7: Lists, Dictionaries and Strings

The human animal differs from the lesser primates in his passion for lists of “Ten Best”
– H. Allen Smith

Textbook

• Strongly Recommended Exercises
  – The Python Workbook:
    • Lists: 109, 110, 118 and 127
    • Dictionaries: 128, 129, and 138
    • Strings: 115, 116, 132, 133 and 135

• Recommended Exercises
  – The Python Workbook:
    • Lists: 105, 107, 108, 117, 119 and 121
    • Dictionaries: 139 and 140
    • Strings: 122, 123, 124, 130 and 131
Textbook

• Recommended Readings
  – Starting Out with Python:
    • Lists: Chapter 8 (2nd Ed.) / Chapter 7 (3rd Ed. and 4th Ed.)
    • Dictionaries: Section 10.1 (2nd Ed.) / Section 9.1 (3rd Ed. and 4th Ed.)
    • Strings: Chapter 9 (2nd Ed.) / Chapter 8 (3rd Ed. And 4th Ed.)

Lists

• Consider the following problem
  – Write a program that reads the high and low temperature of each day for the past year
  – Once the data is read, compute
    • Hottest day, Coldest day
    • Identify heat waves, extended cold periods
    • Determine last day of frost in spring, first day of frost in fall
    • Compute average and median temperature
    • Graph the data
What is a List?

- A collection of values
  - Values
    - May all have the same type, or
    - May have different types
  - Each item is referred to as an element
- Each element has an index
  - Unique integer identifying its position in the list
- A list is one type of data structure
  - A mechanism for organizing related data

Creating a List

- Created like other variables
  - Values are comma separated inside square brackets
  - Examples:

```python
low_temps = [1.4, -1.8, 0.7, 0.9, 1.2, -2.2, -0.3]
names = ["Ben"]
stuff = [1, "ICT", 3.14]
empty = []
```
**Accessing Elements**

- Each list element has a unique index
  - Values range from 0 to length of the list - 1
- To access one element, use the name of the list, followed by the index of that element in square brackets
  - Use this one element just like any other variable

**Changing Elements**

- Individual elements in a list can be changed without impacting the rest of the list

```python
stuff = [1, "ICT", 3.14]
stuff[1] = "Hello"
print(stuff)
stuff[2] = "World"
print(stuff)
```
Loops and Lists

• A for loop iterates over the values in a list
  – List can be created by the range function
  – List can be created by any other means
• Consider the following loop:

```python
stuff = [1, "ICT", 3.14]
for item in stuff:
    print(item)
```

Length of a List

• When a list is initially created, we know its length
  – Adding / removing elements from the list will change its length
  – New length can be determined using the `len` function in the standard library
• Examples:
  • `len([0.69, 3.14, -16.0])` returns 3
  • `len([])` returns 0
Loops and Lists

• Sometimes we need a loop where the control variable varies over the indices rather than the values

```python
stuff = [1, "ICT", 3.14]
for i in range(0, len(stuff))
    print(stuff[i])
```

Adding Elements

• Several methods are defined on lists
  – Use the name of the list you want to work with
  – Follow it by a dot
  – Use the name of the method
  – Provide any required parameters

• Elements are added with append

```python
stuff = [1,"ICT"]
stuff.append(3.14)
print(stuff)
```
Inserting New Elements

• Append allowed us to add an element to the end of a list
  – What if we want to insert an item in the middle of the list?

Searching

• Use `in` to check if an item is present in a list
  – 2 in [1, 2, 3, 4, 5] evaluates to True
  – 8 in [1, 2, 3, 4, 5] evaluates to False

• Use `index` to determine where it is in the list
  – [11, 12, 13, 14].index(12) evaluates to 1
  – [11, 12, 13, 14].index(8) results in a Value Error
Removing

• How can we remove an item from a list?
  – Use the remove method
    • Removes the first occurrence of the item
    • Subsequent identical items remain in the list
    • Item must exist or a Value Error will occur

```python
x = [1, 2, 1, 3, 4, 2, 1]
x.remove(1)
print(x)
```

Removing

• What if we want to remove all occurrences of an item from a list?
Removing

- What if we know the index of the item we want to remove?
  - Use `pop`
  - With no parameters: Removes last item
  - With one parameter: Removes item at the index specified
  - Returns the item that is removed

Example

- Compute the median of a list of values entered by the user
  - User will enter an unknown number of values
  - A blank line will be used to indicate that no additional values will be entered
  - If the list has an odd number of elements
    - Median is the middle value
  - If the list has an even number of elements
    - Median is average of the two middle values
Design

Sorting

• How do we put things into order?
Selection Sort

Insertion Sort
Sorting

- Sorting is an important task
  - Needed when working with large data sets
  - Frequently occurs as part of other algorithms
- Sorting has been studied extensively
  - Many algorithms, some of which are quite complex
  - Selection Sort, Insertion Sort and Bubble Sort
    - Relatively easy algorithms
    - Poor performance for large data sets
Sorting in Python

• Python makes sorting a list easy
  – Use the sorted function
    • Takes one parameter which is an unsorted list
    • Returns a new list sorted into increasing order
  – Use the sort method
    • Invoked on a list using dot notation
    • Does not require any parameters
    • Modifies the list, sorting it into ascending order

Example

• Compute the median of a list of values entered by the user
Other List Operations

• Concatenation
  – Joins two lists
  – Performed using the + operator

• Slicing
  – Extracts a portion of a list
  – Performed using : operator
  – Forms
    • ListName[first:last]
    • ListName[first:last:increment]

More Dimensions

• All of the lists we have used so far have been one-dimensional
• We can add a second dimension by making each element in a list another list

myList = []
myList.append([1,2])
myList.append([3,4])
What Are 2D Lists Used For?

• Images
  – Each element stores a color
• Tables / Spreadsheets
  – Each element stores a value
• Game boards
  – Each element in the list records the piece, if any, that occupies the space
  – Can be used to implement Tic Tac Toe, Chess, Checkers, Boggle, Scrabble, …

Example: Boggle

• Generate a random board for Boggle
  – 4x4 board
  – Store the board in a 2D list
  – Each space on the board contains one randomly selected letter
  – Display the board
  – Sample Board:

```
S N K O
V R E R
I D I N
N E G U
```
Example: Boggle

Tuples

• Similar to lists, but
  - length cannot be changed
  - Items cannot be assigned individually
  - () empty tuple, (3,) length one tuple

aTuple = (1,"ICT",3.14)
From Lists to Dictionaries

• Consider the following problem
  – Many cities in Alberta
  – Want to have a list that contains the populations
  – Need to be able to look up population by city

Dictionaries

• Dictionary: A collection of values
  – Each element in a list has an index
    • A unique integer, starting from 0
  – Dictionaries allow us to extend this idea
    • Each value in the dictionary has a unique identifier associated to it
      – Referred to as a key
      – Can be a string or a number
    • Starting in Python 3.7, the key-value pairs in a dictionary are always insertion ordered
Dictionary Example

• Create a dictionary that describes the population of several Alberta cities

Adding to a Dictionary

• What if we want to add more cities to our dictionary later in the program?
Removing Items

• Remove one item
  – Use pop
    • Example: cities.pop("Calgary")

• Remove all items
  – Use clear method
  – Example: cities.clear()

Dictionary Methods

• Want a list of the keys in a dictionary?
  – Use dictionary_name.keys()
  – Example:

    for i in cities.keys():
        print(cities[i],"people live in",i)
Dictionary Methods

• Want a list of values in a dictionary?
  – Use dictionary_name.values()
• Example: Compute the total population of all of the cities

Dictionaries Example

• Consider the following problem
  – We have a list of values
  – Want to determine the mode for the list
    • Mode is defined to be the most frequently occurring value
    • A list may have more than one mode
Dictionaries Example

Dictionaries Summary

• Dictionaries
  – Hold a collection of values
  – Unordered
  – Each element is a key value pair
    • Easy to lookup the value associated with each key
Strings

• Strings
  – A collection of characters
  – Numerous methods available for manipulating strings
    • upper
    • lower
    • swapcase
    • rjust
    • …

Strings

• Strings provide additional methods for searching, separating, etc.
  – Processing input from the user is challenging
    • Anything could be entered
    • Generally want our program to handle this nicely
    • Common to expend significant effort processing input before it is passed to the rest of the program
Searching

• The find method searches a string for a substring

```python
s = "Hello World!"
print(s.find("ll"))
print(s.find("o"))
print(s.find("o",5))
print(s.find("Wor",0,6))
```

Separating

• Use split
  – Returns a list of strings
  – Splits the string at each separator character that is encountered

```python
s = "This is a test string"
list = s.split(" ")
for i in list:
    print(i)
```
Extracting Characters

• Characters in a string can be accessed by index
  – Enclose index of single character in square brackets
  – Use : to form a slice

s = "Hello World!"
print(s[3])
print(s[6:])

String Example: Validating a Password

• Write a function that determines if a password is (somewhat) secure
  – Has at least 7 characters
  – Contains at least one upper case letter
  – Contains at least one lower case letter
  – Contains at least one numeric digit
String Example: Validating a Password

Functions Involving Strings, Lists and Dictionaries

- Lists, Dictionaries & Strings
  - Can be passed as parameters
  - Can be returned as results

- Care must be taken to avoid inadvertently modifying a list or dictionary (not string) inside a function
Functions Involving Lists and Dictionaries

Mutable vs. Immutable Types

• In python, every variable is an object
  – Consists of
    • a pointer to some memory
    • value(s) stored in that memory
  – The location that the pointer points to can change
  – For mutable types, the values stored in memory can also change
  – Values stored in memory can not change for immutable types
Mutable vs. Immutable Types

• What happens when a new value is assigned to a variable storing an immutable type?

Mutable vs. Immutable Types

• What happens when we change a value in a list (a mutable data type)?
Mutable vs. Immutable Types

• Examples of Immutable Types
  – Integer, Float
  – String
  – Boolean
  – …

• Examples of Mutable Types
  – Lists
  – Dictionaries
  – …

Mutable vs. Immutable Types Review

• What happens when you change the value of a variable with immutable type?

• What happens when you change a variable with mutable type?
Mutable vs. Immutable Types Review

• Which types are immutable?

• Which types are mutable?

• Why are some types immutable and other types mutable?

Key Points

• Mutable vs. Immutable Types
  – Memory at the end of the arrow doesn’t change for immutable types
  – Changing the value of a variable with immutable type causes it to point to a different piece of memory
  – Changing a variable with immutable type in the called scope will not change the value of the variable in the calling scope
Wrapping Up

• Data structures allow us to organize larger amounts of information
  – Lists hold many values (ordered)
    • May have same type or may have different types
    • Each element has a unique integer index, starting from zero
  – Dictionaries hold many values
    • Each element consists of a key-value pair
    • Items can be looked up by key
    • Unordered data structure

• Strings help us organize character data
  – Provide mechanisms for searching and splitting strings
    • Can be used to validate user input

• Lists, dictionaries and strings can be passed to and returned from functions
  – Strings are immutable
  – Lists and dictionaries are mutable
Where Are We Going?

• Data structures allow us to manage larger amounts of data in a reasonable way
  – Larger amounts of data typically come from disk
    • Too much to enter by hand
  – How do we load data from files?
  – How do we save data in files?
  – How do we handle errors?