

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

1

Textbook

- Strongly Recommended Exercises
 - The Python Workbook, 3rd Edition:
 - Lists: 128, 129, 140 and 150
 - Dictionaries: 153, 154, and 166
 - Strings: 135, 136, 157, 158 and 160
 - The Python Workbook, 2nd Edition:
 - Lists: 115, 116, 125 and 135
 - Dictionaries: 136, 137, and 146
 - Strings: 122, 123, 140, 141 and 143

2

Textbook

- Recommended Exercises

- The Python Workbook, 3rd Edition:
 - Lists: 124, 126, 127, 139, 141, and 143
 - Dictionaries: 167 and 168
 - Strings: 144, 145, 146, 147, 155, and 156
- The Python Workbook, 2nd Edition:
 - Lists: 111, 113, 114, 124, 126, and 128
 - Dictionaries: 147 and 148
 - Strings: 129, 130, 131, 132, 138, and 139

- Recommended Readings

- Starting Out with Python, 2nd or 3rd Edition: Chapters 5 and 6

3

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

4

What is a List?

- A collection of items
 - Items
 - 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

5

Creating a List

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

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

6

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

7

Changing Elements

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

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

8

Loops and Lists

- A for loop iterates over a collection of values
 - The collection can be created by the range function
 - The collection can be a list
- Consider the following loop:

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

9

Length of a List

- When a list is initially created, we know its length
 - Adding / removing elements from the list will change its length
 - Current length can be determined using the `len` function
 - Examples:
 - `len([0.69, 3.14, -16.0])` returns 3
 - `len([])` returns 0

10

Loops and Lists

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

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

11

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

```
stuff = [1, "ICT"]
stuff.append(3.14)
print(stuff)
```

12

Inserting New Elements

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

13

Searching

- Use `in` to check if an element is present in a list
- Use `index` to determine where it is in the list

```
items = []

...    # Add some elements to items

if 0 in items:
    zero_index = items.index(0)
    ...    # Do other interesting things
else:
    print("There wasn't a 0 in items.")
```

14

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 ValueError will occur

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

15

Removing

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

16

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

17

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

18

Design

19

Sorting

- How do we put things into order?

20

Selection Sort

21

Insertion Sort

22

Bubble Sort

23

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

24

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

25

Example

- Compute the median of a list of values entered by the user

26

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]`

27

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

```
data = []  
data.append([1,2])  
data.append([3,4])
```

28

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, ...

29

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

31

Example: Boggle

32

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

33

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 stored in the order in which they were inserted

34

Dictionary Example

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

35

Adding to a Dictionary

- What if we want to add more cities to our dictionary later in the program?

36

Removing Items

- Remove one item
 - Use `pop`
 - Example: `cities.pop("Calgary")`
- Remove all items
 - Use `clear` method
 - Example: `cities.clear()`

37

Dictionary Methods

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

```
for city in cities.keys():  
    print(cities[city], "people live in", city)
```

38

Dictionary Methods

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

39

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

40

Dictionaries Example

41

Dictionaries Summary

- Dictionaries
 - Hold a collection of values
 - Each element is a key value pair
 - Easy to lookup the value associated with each key
 - New key value pairs can only be added at the end
 - No ability to insert in the middle of the collection
 - Key value pairs can be removed

42

Strings

- Strings
 - A collection of characters
 - Numerous methods available for manipulating strings
 - upper
 - lower
 - swapcase
 - rjust
 - ...

43

Strings

- Strings include methods for searching, separating, etc.
 - Processing input from the user is challenging
 - Anything could be entered
 - Want our program to handle this nicely
 - Common to expend significant effort processing input before it is passed to the rest of the program

44

Searching

- The find method searches a string for a substring

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

45

Separating

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

```
s = "This is a test string"
words = s.split(" ")
for w in words:
    print(w)
```

46

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:])
```

47

String Example: Validating a Password

- Write a function that determines if a password meets a particular set of constraints
 - 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

48

String Example: Validating a Password

49

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

50

Functions Involving Lists and Dictionaries

51

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

52

Mutable vs. Immutable Types

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

53

Mutable vs. Immutable Types

- What happens when we change a value in a list (a mutable data type)?

54

Mutable vs. Immutable Types

- Examples of Immutable Types
 - Integer, Float
 - String
 - Boolean
 - ...
- Examples of Mutable Types
 - Lists
 - Dictionaries
 - ...

55

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?

56

Mutable vs. Immutable Types Review

- Which types are immutable?
- Which types are mutable?
- Why are some types immutable and other types mutable?

57

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 parameter variable with immutable type in the called scope will not change the value of the argument in the calling scope

58

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
 - Cannot insert into the middle of the collection

59

Wrapping Up

- 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

60

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?

61