Chapter 5:
Lists and Dictionaries
In this chapter, you will learn how to:

- Create, Index and Slice a **LIST**
- Add and Delete elements of a LIST
- Use List Methods to Append, Sort and Reverse a List
- Use Nested Sequences to represent even more Complex Information
- Use Dictionaries to Work with Pairs of Data
- Add and Delete Dictionary Items
Creating a List

- **Lists work just like tuples**

- **Lists are mutable**

- **To assign elements to a List:**
  
  inventory = [“sword”, “armor”, “shield”, “healing potion”]
List

Printing List elements

- Print elements in List, “inventory”

inventory = [“sword”, “armor”, “shield”, “healing potion”]

for item in inventory:
    print item

raw_input (“\n\nPress the enter key to exit.”)

Displays:

<table>
<thead>
<tr>
<th>sword</th>
<th>armor</th>
<th>shield</th>
<th>healing potion</th>
</tr>
</thead>
</table>
List

Use the `len()` Function with Lists

- Get length of a List, “inventory”:
  - `inventory = [“sword”, “armor”, “shield”, “healing potion”]`
  - `isize = len (inventory)`
    - Function returns number of elements in list variable, `inventory`
  - print “You have”, `len(inventory)`, “items in your possession”

Displays:

You have 4 items in your possession
List

Test for membership in a List

- Use the `in` operator with List
- `inventory = ["sword", "armor", "shield", "healing potion"]`

- `if "shield" in inventory:`
  - `print "You are protected by Blue Cross and Blue Shield"`
List

Indexing Lists

print inventory[0]
print inventory[1]
print inventory[-1]

raw_input(“\nPress enter to exit”)

Program Output:
sword
armor
shield

Indexing:
Restricted to one element at a time
# List

## Slicing List

### Slice inventory:

<table>
<thead>
<tr>
<th>Start Position</th>
<th>End Position</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>0:3</td>
<td></td>
<td>['sword', 'armor', 'shield']</td>
</tr>
<tr>
<td>1:3</td>
<td></td>
<td>['armor', 'shield']</td>
</tr>
<tr>
<td>-3:-1</td>
<td></td>
<td>['sword', 'armor']</td>
</tr>
</tbody>
</table>

![Image of list slicing](image.png)
Concatenating Lists

- Concatenate two Lists: chest & inventory
- chest = [“gold”, “gem”]
- inventory = [“sword”, “armor”, “shield”, “healing potion”]
- inventory = inventory + chest
- print inventory

Display


NOTE: We can only concatenate sequences of the same type
Lists

mutability

- Lists are Mutable
  - Can Assign a New List Element by Index
  - Assign a New List Slice
  - Delete a List Element
Lists

Assign New List Element by Index

Replace “sword” with “crossbow”

inventory = [“sword”, “armor”, “shield”, “healing potion”]

inventory [0] = “crossbow”

print “Your new inventory is:
”, inventory
raw_input(“
Press Enter key to exit”)

Output

Your new inventory is:
[‘crossbow’, ‘armor’, ‘shield’, ‘healing potion’]
Lists

Assign New List Slice

Let’s assign the list [“future telling”] to the slice inventory[4:6]

```python
inventory = [“crossbow”, “armor”, “shield”, “healing potion”]
print (“Your new inventory is:
”, inventory)
raw_input(“Press Enter key to exit”)```

Output

Your new inventory is:
[‘crossbow’, ‘armor’, ‘future healing’]
Lists

Deleting a List element

- Use the **del** command to delete List element

```python
inventory = ["crossbow", "armor", "future healing"]

del inventory[1]

print "Your new inventory is:\n", inventory
raw_input("\nPress Enter key to exit")
```

**Output**

```
Your new inventory is:
[‘crossbow’, ‘future healing’]
```
Lists

Deleting a List slice

- Use the del command to delete List slice

```python
inventory = ["crossbow", "shield", "armor", "future healing"]

del inventory[ :2]

print "Your new inventory is:\n", inventory
raw_input("\nPress Enter key to exit")
```

Output

```
Your new inventory is:
[‘armor’, ‘future healing’]
```
Lists
Methods

- Using List Methods
  - Add value to end of list
  - Sort elements, smallest value first
  - Reverse order of list
  - Locate the first position of a value in a list
  - Remove first occurrence of a value from the list

- List vs. Tuples

- Nested Sequences
  - Lists, Tuples
Using List Methods

**Append Function**

- Append a new value to a given list
  - Note Append means tack to end

```python
scores = ['92', '94', '80', '94']
final_exam = '96'
scores.append(final_exam)
print(scores)
raw_input("\nPress Enter key to exit")
```

**Output:**

```
['92', '94', '80', '94', '96']
```
Using List Methods

Index Function

- Locate the first position of a value within a list

  - scores = [“92”, “94”, “80”, “94”, “96”]
  - print scores.index(“94”)
Delete first occurrence of value, 94, from the list

- scores = [“92”, “94”, “80”, “94”, “96”]
- scores.remove(“94”)
- print scores
- raw_input(“\nPress Enter key to exit”)

Displays

[‘92’, ‘80’, ‘94’, ‘96’]
The sort function sorts list values from lowest to highest

- scores = ["92", "94", "80","96"]
- scores.sort()
- print scores
- raw_input("\nPress Enter key to exit")

Displays

[‘80’,’92’, ‘94’,’96’]
Using List Methods

reverse Function

- Reverse the current order of a list

  - `scores = ['80','92', '94','96']`
  - `scores.reverse()`
  - `print scores`

Displays

`['96','94', '92','80']`
Using List Methods

**insert** Function

- Insert a value in any location, \( n \), of the list
  - \( \text{scores} = ['80', '92', '94', '96'] \)
  - Let’s insert extra credit (“90”) in position 1
    - \( \text{scores.insert}(1, "90") \)
    - print scores

Displays

\[ ['80', '90', '92', '94', '96'] \]
List vs. Tuples

- Tuples have faster access than Lists

- Tuples are perfect for storing constants
  - Tuples are immutable

- Lists are mutable
  - Easy to modify sequence
  - Flexible to use
Nested Sequences

- Lists can contain
  - Other Lists
  - Tuples

- Tuples can contain
  - Lists
  - Other Tuples
Create Nested Sequence

- `category = [“first”, (“second”, “third”), [“fourth”, “fifth”, “sixth”]]`
- `print category`

Displays

Nested Sequences

Caution

- Create nested sequences with uniform pattern!
- \( \text{score} = [\text{“Larry”, (“Quiz”, 80), (“H-Work”, 85), (“Test”, 95), (“Final”, 90)}] \)
Nested Sequences

Accessing Nested Elements

- scores = [ (“Moe”, 1000), (“Larry”, 1500), (“Curly”, 3000)]
- print scores[0]
- print scores[1]
- print scores[0][1]
- print scores[1][0]

Displays

(‘Moe’, 1000)
(‘Larry’, 1500)
1000
Larry
Unpacking a Sequence

- Count the number of elements in the sequence
- Assign each element to independent variable names

```
name, score = ("Moe", 1000)
print name
print score
```

Displays

```
Moe
1000
```
Setting Up Your Program

Choices/Options

- Execute Different Program Segments based on user choices/options

- **Step #1:** Display Choices/Options
- **Step #2:** Code to Read User Input ~ `raw_input(...)`
- **Step #3:** Create: if – elif – else structure:
  - use “if” and “elif” conditional statements to test each choice/option
  - use the “else” for the default condition
Shared References

Variable Names

- `language = “python”`
Shared References

mike = ["khakis", "shirt", "jacket"]
Shared References

Pointers

mike = [“khakis”, “shirt”, “jacket”]

tim = mike
mr_porter = mike

Tim and mr_porter refer to the same address as mike
mike = [“khakis”, “shirt”, “jacket”]

tim = mike
mr_porter = mike
print mike
print tim
print mr_porter

Displays

[‘khakis’, ‘shirt’, ‘jacket’]

[‘khakis’, ‘shirt’, ‘jacket’]

[‘khakis’, ‘shirt’, ‘jacket’]
Shared References

Pointers

mike = [“khakis”, “shirt”, “jacket”]


Any change made to list using one variable affects the other two
print mike
print tim
print mr_porter

Displays

['khakis', 'shirt', 'suit']
['khakis', 'shirt', 'suit']
['khakis', 'shirt', 'suit']
Shared References

Copy

- mike = ["khakis", "shirt", "jacket"]
- tim = mike[:]
- print mike
- print tim

Displays

[‘khakis’, ‘shirt’, ‘suit’]

[‘khakis’, ‘shirt’, ‘jacket’]
**Dictionaries**

**Create**

Dictionary name =

{“key #1” : “key value #1”,
 “key #2” : “key value #2”,
 “key #n” : “key value #n”}

gEEK =

{ “CS 10061” : “Intro Computer Programming”,
 “404” : “Clueless. Page not found”}
Dictionaries

Accessing

- Use the **key** to retrieve the corresponding **value**

```python
geek = { "CS 10061" : "Intro Computer Programming",
        "404"     : "Clueless. Page not found"}
```

- ```print geek["CS 10061"]```  
- ```print geek["404"]```  

**OUTPUT**

```
'Intro Computer Programming',

'Clueless. Page not found'
```

---

**CAUTION:** Check if a key exist prior to retrieving corresponding value
Dictionaries

Testing for a key

- **get** Method Construct
- Test if “Link Rot” is a key in *geek*:

```python
geek.get("Link Rot", "I have no idea")
```

Dictionary → Method → Search key → Message returned, if search key does not exist

default

None
Dictionaries

Adding a key-Value Pair

Add a term-definition pair
- `geek [term] = definition`

term = raw_input(“What term do you want to add? “)

If term not in geek:
  definition = raw_input(“Enter the definition for term: ”)
  geek[term] = definition
else
  print “Term already exists”
Dictionaries

Replacing a key-Value Pair

- Add a term-definition pair
- `geek[term] = definition`

```
term = raw_input("What term do you want to redefine? ")

If term in geek:
    definition = raw_input("Enter the new definition : ")
    geek[term] = definition
else
    print "Term does not exist"
```
Dictionaries

Deleting a key-Value Pair

- Delete a term-definition pair
- `del geek [term]`

```python
term = raw_input("What term do you want to delete? ")
If term in geek:
    del geek[term]
else:
    print "Term does not exist"
```

Now let’s run the geek translator program
Dictionaries

Summary

- Key must be unique

- Key:
  - String, Number, or a Tuple

- Key must be immutable

- Values need not be unique