CSCI A201/A597

Introduction to Programming I

Lecture 10 draft

Fall 2018
A201 / A597 Test 02 – review

please read thoroughly
the Test 02 Review page
linked to the main
A201 / A597 page!

https://homes.sice.indiana.edu/classes/fall2018/csci/a201-mitja/2018/tests/a201-test02-review.html
review: Using Nested Sequences

- **Nested Sequence:**
  A sequence inside another sequence

- A list can contain lists, tuples (and dictionaries)

- A tuple can contain tuples, lists (and dictionaries)

- (and dictionaries can contain tuples, lists, etc.)
review: Creating Nested Sequences

```python
>>> scores = [("Jo", 1000), ("Jill", 1500), ("Thursday", 3000)]
>>> print(scores)
[('Jo', 1000), ('Jill', 1500), ('Thursday', 3000)]
```

- **scores** is a nested sequence
- **scores** is a list of tuples
- **scores** has three elements, each of which is a tuple
review: Accessing Nested Elements

```python
>>> scores = ["Jo", 1000), ("Jill", 1500), ("Thursday", 3000)]
>>> print (scores[2])
('Thursday', 3000)
>>> print (scores[2][0])
Thursday
```

- **scores[2]** is the element of the list at position 2
- **scores[2][0]** is the element at position 0 of scores[2]
Unpacking a Sequence

```python
>>> name, score = ("Next", 175)
>>> print (name)
Next
>>> print (score)
175
```

- **Sequence unpacking:**
  Automatically accessing each element of a sequence

- The tuple is unpacked as result of assignment statement
Accessing Elements of a Nested Sequence

for entry in scores:
    score, name = entry
    print (name, "\t", score)

- **entry** is an element of **scores**
- **Assignment statement unpacks** **entry**
- **score** is assigned first element of **entry**
- **name** is assigned second element of **entry**
Appending Elements to a Nested Sequence

```python
element = (score, name)
scores.append(entry)
```

- `.append()` method works for any list, including a list of sequences
- New tuple entry is created
- `entry` is appended to list `scores` as last element
• a variable, and the object that is referred by that variable
• the variable **language** refers to computer memory where "Python" is stored.
review: Shared References (continued)

• variables *don't store objects*:

  variables *refer to objects*

• Shared Reference:
  A reference to an object, which has at least one other reference to it

• Shared references have significance for mutable objects
review: Shared References (continued)

- example from a book listed in references (Michael Dawson: Python Programming for the Absolute Beginner, 3rd Ed.,)

- A single object has three references to it.
- **mike, mr_dawson** and **honey** all refer to same single list.
>>> mike = ["khakis", "dress shirt", "jacket"]
>>> mr_dawson = mike
>>> honey = mike
>>> print mike
["khakis", 'dress shirt', 'jacket']

>>> print mr_dawson
["khakis", 'dress shirt', 'jacket']

>>> print honey
["khakis", 'dress shirt', 'jacket']

• example from a book listed in references (Michael Dawson: Python Programming for the Absolute Beginner, 3rd Ed.,)

• the above variables refer to same single list
>>> honey[2] = "red sweater"
>>> print honey
['khakis', 'dress shirt', 'red sweater']

>>> print mike
['khakis', 'dress shirt', 'red sweater']

>>> print mr_dawson
['khakis', 'dress shirt', 'red sweater']

• example from a book listed in references (Michael Dawson: Python Programming for the Absolute Beginner, 3rd Ed.,):

• change to list through one variable reflects change for all variables because there is only one list
review: Shared References (continued)

```python
>>> mike = ["khakis", "dress shirt", "jacket"]
>>> honey = mike[:]
>>> honey[2] = "red sweater"
>>> print honey
['khakis', 'dress shirt', 'red sweater']

>>> print mike
['khakis', 'dress shirt', 'jacket']
```

• List slicing can **create a new copy** of a list and avoid shared references
a new type: Dictionaries

- Dictionary:
  A mutable collection of key-value pairs
  Just like tuple and list, dictionary is a built-in Python type
- Unlike tuples and lists, dictionaries don't organize data into sequences, instead they organize data into pairs

- Works like actual dictionary where you look up one thing to get another:
- in a Python dictionary, you look up a key to get a value
The Geek Translator Program

- example Python program from a book listed in references (Michael Dawson: Python Programming for the Absolute Beginner, 3rd Ed.,)
- Sample run of the Geek Translator program
- Geek terms and definitions are accessed with a dictionary.
- Source code and book listed in:
  https://homes.sice.indiana.edu/classes/fall2018/csci/a201-mitja/2018/resources/references.html
Creating Dictionaries

greek = {
"404" : "clueless.",
"Uninstalled" : "being fired."}

- example from the "Geek Translator" program:
- creates new dictionary called **greek**
- **greek** has two entries or items
- Each **item** is made up of a **key** and a **value**
- **404** is a key of one item;
  use it to look up value "**clueless**."
- Create dictionary by pairing values with colon,
  separated by commas, surrounded by curly braces
Using a Key to Retrieve a Value

```python
>>> geek["404"]
'clueless.'
>>> geek["Uninstalled"]
'being fired.'
```

- Use key as index to get value
- Cannot use value as index to get key
- Using non-existent key as index produces error
- Dictionaries don't have position numbers – **no order**

- (that's because the underlying data structure implementing dictionaries is a **hash table** – it's one of the most efficient data structures there is... but it does not preserve any order!)
Testing for a Key with the in Operator

example from the "Geek Translator" program:

```python
>>> if "Dancing Baloney" in geek:
    print "I know what Dancing Baloney is."
else:
    print "I have no idea what Dancing Baloney is."
```

I have no idea what Dancing Baloney is.

•

•Use the **in** operator to test for key
•Condition is True if key exists in dictionary, False otherwise
•**in** operator *can't* be used to test for dictionary *values*
The Dictionary get() Method

```python
>>> geek.get("404")
'clueless.'
>>> geek.get("Dancing Baloney")
None
>>> geek.get("Dancing Baloney", "I have no idea.")
'I have no idea.'
```

- example from the "Geek Translator" program:
- `.get()` is used for retrieving value based on key
- Has built-in "safety" net for handling non-existent key
  - If key exists, returns associated value
  - If key doesn't exist, returns a default value
Adding a Key-Value Pair

example from the "Geek Translator" program:

geek["Link Rot"] = "process by which web page links become obsolete."

• Dictionaries are mutable

• Add item by assigning value to dictionary indexed by key

• Overwrites current entry if key already exists in dictionary
Deleting a Key-Value Pair

time example from the "Geek Translator" program:

del geek["404"]

• Removes key-value pair if key exists

• Generates error if key doesn't exist
Table 5.2: Selected Dictionary Methods

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>get(key, [default])</code></td>
<td>Returns the value of <code>key</code>. If <code>key</code> doesn't exist, then the optional <code>default</code> is returned. If <code>key</code> doesn't exist and <code>default</code> isn't specified, then <code>None</code> is returned.</td>
</tr>
<tr>
<td><code>keys()</code></td>
<td>Returns a view of all the keys in a dictionary.</td>
</tr>
<tr>
<td><code>values()</code></td>
<td>Returns a view of all the values in a dictionary.</td>
</tr>
<tr>
<td><code>items()</code></td>
<td>Returns a view of all the items in a dictionary. Each item is a two-element tuple, where the first element is a key and the second element is the key’s value.</td>
</tr>
</tbody>
</table>
Dictionary Requirements

• **Keys**
  • Must be unique
  • Must be immutable

• **Values**
  • Can be mutable or immutable
  • Doesn't have to be unique
Summary of reviewed topics:

• A nested sequence is a sequence inside another sequence

• Sequence unpacking is the process of automatically accessing each element of a sequence

• A shared reference is a reference to an object, which has at least one other reference to it
Summary of new topics: Dictionary

- A dictionary is a mutable collection of key-value pairs
- In a dictionary, an item is a key-value pair
- In a dictionary, a key is an object used to look up another object
- In a dictionary, a value is an object that is returned when its corresponding key is looked up
- The in operator can be used to test if a dictionary contains a specific key
Summary of new topics: Dictionary (continued)

- A dictionary can't contain multiple items with the same key
- A dictionary can contain multiple items with the same value
- Dictionary keys must be immutable
- Dictionary values can be mutable