CMSC201
Computer Science I for Majors

Lecture 17 – Dictionaries

Last Class We Covered

- File I/O
  - Input
    - Reading from a file
    - **`read()`**, **`readline()`**, **`readlines()`**, **`for`** loops
  - Output
    - Writing to a file
- Manipulating strings (and lists of strings)
  - **`split()`**, **`join()`**

Any Questions from Last Time?

Announcement – Survey #2

- Available now on Blackboard
- Due by Sunday, November 13, at midnight
  - Check completion under “My Grades”
- Some statistics (from Fall 2015):
  - If they had taken the surveys…
    - 9 students would have gotten an A instead of a B
    - 4 students would have gotten a B instead of a C
    - 9 students would have gotten a C instead of a D

Today's Objectives

- Construct dictionaries and access entries in those dictionaries
- Use methods to manipulate dictionaries
- Decide whether a list or a dictionary is an appropriate data structure for a given application

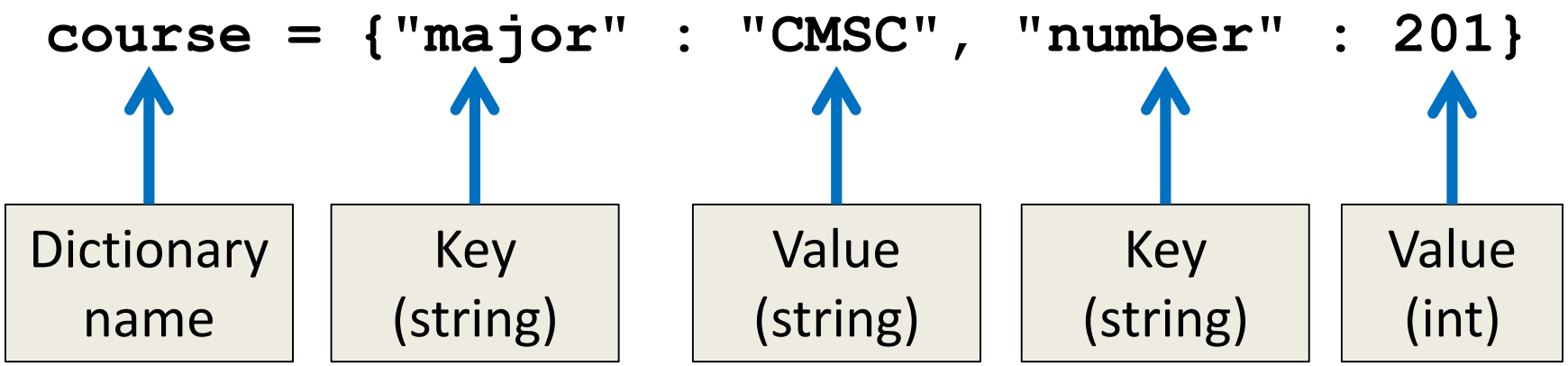
Organization

- Information in a list is organized how?
  - By order
- Information in a dictionary is organized
  - By ***association***

- Python dictionaries associate a set of ***keys*** with corresponding data ***values***

Keys and Values

- A dictionary is a set of “keys” (terms), each pointing to their own “values” (meanings)



Dictionary Keys

- Think of a dictionary as an unordered set of ***key:value*** pairs

- Dictionary keys must be ***unique***
  - A key in a dictionary is like an index in a list
  - Python must know exactly which value you want

- Keys can be of any data type
  - As long as it is ***immutable***

Dictionary Values

- Dictionary keys have many rules, but the values do not have many restrictions

- They do not have to be unique
  - Why?

We can have duplicate values in a list, but indexes must be unique

- They can be mutable or immutable
  - Why?

Since they don't need to be unique, we can change them without restriction

Creating Dictionaries

Creating Dictionaries

- There are three main ways to create a dictionary in Python:
  1. Construct a python dictionary (using the curly braces syntax)
  2. Construct a dictionary from a list of key, value ***pairs***
  3. Construct a dictionary from two lists

Creating Dictionaries (Curly Braces)

- The empty dictionary is written as two curly braces containing nothing

```
dict1 = {}
```

- To create a dictionary, use curly braces, and a colon (:) to separate keys from their value

```
dict2 = {"name" : "Maya", "age" : 7}
```

Creating Dictionaries

```
dict3 = [('a', 'apple')]
print (dict3, type(dict3))
```

Is this a dictionary?

```
[('a', 'apple')] <class 'list'>
```

Must use curly braces { } to define a dictionary

Creating Dictionaries

```
dict4 = {('a', 'apple')}
print (dict4, type(dict4))
```

Is this a dictionary?

```
{('a', 'apple')} <class 'set'>
```

Must use a colon (:) between items, not a comma

Creating Dictionaries

```
dict5 = {('a' : 'apple')}
print (dict5, type(dict5))
```

Is this a dictionary?

```
{'a': 'apple'} <class 'dict'>
```

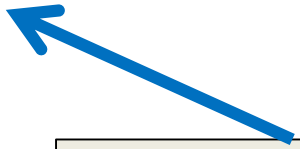
Hooray!

Creating Dictionaries (From a List)

- To cast a list as a dictionary, you use **`dict()`**

```
myPantry = [(5, 'candy'),
 (15, 'cookies'),
 (23, 'ice cream')]

# cast to a dictionary
myDict = dict(myPantry)
```



Must be key:value pairs

Dictionary Operations

Dictionary Operations

- Dictionaries are probably most similar to a list

- You can do a number of operations:
  - Access a key's value
  - Update a key's value
  - Add new key:value pairs
  - Delete key:value pairs

Accessing Values

- To access dictionary elements, you use the square brackets and the key to obtain its value

```
dogBreeds = {"A" : "Akita", "B" : "Basenji",
             "C" : "Chesapeake Bay Retriever"}
print("dogBreeds at C:", dogBreeds["C"])
print("dogBreeds at B:", dogBreeds["B"])

Output:
dogBreeds at C: Chesapeake Bay Retriever
dogBreeds at B: Basenji
```

Updating Values

- To update dictionary elements, you use the square brackets and the key to indicate which value you would like to update

```
dogBreeds["B"] = "Beagle"
print(dogBreeds)

Output:
{'C': 'Chesapeake Bay Retriever',
'B': 'Beagle', 'A': 'Akita'}
```

Why are these out of order?

Dictionaries organize by ***association***, not by order

Adding New Key:Value Pairs

- To add new values, we don't need to use **`append()`** – we simply state the key and value we want to use, with square brackets

```
dogBreeds["D"] = "Dunker"
dogBreeds["E"] = "Eurasier"
print(dogBreeds)

Output:
{'C': 'Chesapeake Bay Retriever', 'B': 'Beagle',
'A': 'Akita', 'E': 'Eurasier', 'D': 'Dunker'}
```

Deleting Key:Value Pairs

- Key:value pairs must be deleted together; you can't have a key with no value
- To delete a key:value, use the **`del`** keyword and specify the key you want to delete

```
del dogBreeds["D"]
print(dogBreeds)
```

```
Output:
{'C': 'Chesapeake Bay Retriever', 'B': 'Beagle',
'A': 'Akita', 'E': 'Eurasier'}
```

Time for…

LIVECODING!!!

Creating Dictionaries (From Two Lists)

- Here we have two lists
  - Of the same length
  - Contents of each index match up
    - (Tina is Social Work, Pratik is Pre-Med, etc.)

```
names = ["Tina", "Pratik", "Amber"]
major = ["Social Work", "Pre-Med", "Art"]
```

- Write the code to create a dictionary from these

Dictionary Functions and Methods

Functions and Methods

- `len(theDictionary)`
- `str(theDictionary)`
- `type(variable)`
- `theDictionary.get(theKey)`
- `theDictionary.items()`
- `theDictionary.values()`
- `theDictionary.keys()`

Functions

- **`len(theDictionary)`**
  - Gives the length of the dictionary passed in
  - Number of key:value pairs
- **`str(theDictionary)`**
  - Returns a printable string representation
- **`type(variable)`**
  - Returns the type of the passed variable
  - If a dictionary is passed, type returned is <class 'dict'>

Methods

- Methods are functions that are specific to a data type (like **`append()`** or **`lower()`**, etc.)

- **`theDictionary.get(theKey)`**
  - For a key **`theKey`**, returns the associated value
  - If **`theKey`** doesn't exist, returns **`None`**
  - Optionally use a second parameter to return something other than **`None`** if not found
    - **`theDictionary.get(theKey, -1)`**

Methods

- **`theDictionary.items()`**
  - Returns a "view" of the **`theDictionary`**'s contents
  - Need to cast to a list; list is of key:value **tuple** pairs
  - Useful when you want to iterate over a dictionary

```
for item in list(theDictionary.items()):
    # split tuple into key and value
    key, val = item
    print("key:", key)
    print("value:", val)
```

Methods

- **`theDictionary.values()`**
  - Returns a "view" of the **`theDictionary`**'s values
  - Need to cast to a list
- **`theDictionary.keys()`**
  - Returns a "view" of the **`theDictionary`**'s keys
  - Need to cast to a list

- The two lists returned are in the same order
  - (Value at index 0 matches key at index 0, etc.)

When to Use Dictionaries

- Dictionaries are very useful if you have...
  - Data whose order doesn't matter
  - A set of unique keys
    - Words for key, definition or translation for value
    - Postal abbreviations for key, full state name for value
    - Names for key, a list of their game scores for value
  - A need to find things easily and quickly
  - A need to easily add and remove elements

Announcements

- Survey #2 is out
  - Due Sunday, Nov 13 @ 11:59 PM
- Project 1 is due next Wednesday
  - It is much harder than the homeworks
  - No collaboration allowed
  - Start early
  - Think before you code
  - Come to office hours