# Dictionaries

CSSE 120—Rose Hulman Institute of Technology

## Data Collections

- Frequently several individual pieces of data are related
- We can collect them together in one object
- Examples:
  - A list or tuple contains an ordered sequence of items
  - A string contains an ordered sequence of characters
  - A custom object. Example from zellegraphics: A Line object contains two endpoints, a color, and the window in which it is drawn
  - A dictionary (defined soon) contains key-value pairs

## List - review

- an ordered collection of items
- Usually homogeneous (all items of the same type), but Python does not require this
- Access is **by position** (index) in the list

  -
    ```
    >>> animals = ['dog', 'cat', 'cow']
    >>> animals[1]
    'cat'
    >>> animals[1:3]
    ['cat', 'cow']
    >>> animals[1] = ['pig']
    >>> animals
    ['dog', 'pig', 'cow']
    ```

## More list mutations

- Items can be added, removed, or replaced

  -
    ```
    >>> animals = ['dog', 'cat', 'cow']
    >>> animals.append('pig')
    >>> animals
    ['dog', 'cat', 'cow', 'pig']
    >>> animals[1:3] = ['cow', 'cat', 'goat']
    >>> animals
    ['dog', 'cow', 'cat', 'goat', 'pig']
    >>> animals[1:2] = []
    >>> animals
    ['dog', 'cat', 'goat', 'pig']
    ```

**Q1**

## Dictionary

- A collections object in which each item is a key-value pair
- No two items may have the same key
  - So a dictionary is a function (in the mathematical sense)
- Items are not stored in any particular order
- Typically all keys are same type (not required)
- Keys must be immutable (i.e., number, string, tuple)
- Access to items is by key
  - key's purpose is similar to list's index
  - syntax also similar

## Your turn

- Open IDLE and make a quick dict
- Try the following:

```
>>> myDict = {'name':'Dave', 'gpa':3.5}
>>> print myDict
>>> myDict['name']
>>> myDict['gpa']
>>>dir(dict)
```

## Dictionary methods

Assume that there is a dictionary named dict1

- dict1.get(k [,d]) → if k is a **key** in the dictionary return the **value** for that key, else return d. d is an optional parameter
- dict1.has_key(k) → True if dict1 has a **key** k, else False
- dict1.items() → list of dict1's (key, value) pairs, as tuples
- dict1.keys() → list of dict1 's **keys**
- dict1.pop(k [,d] ) → remove **key** and return **value**
- dict1.values() → list of dict1 's **values**
- Checkout **Session16Dictionaries &** open dictionaryMethods.py

## Another dictionary example

-
  ```
  gradeLowestScore = { } # empty dictionary
  gradeLowestScore['A'] = 89.5
  gradeLowestScore['B+'] = 84.5
  gradeLowestScore['B'] = 79.5
  gradeLowestScore['C+'] = 74.5
  gradeLowestScore['C'] = 69.5
  gradeLowestScore['D+'] = 64.5
  gradeLowestScore['D'] = 59.5
  gradeLowestScore['F'] = 0.0
  ```

-
  ```
  difference = gradeLowestScore['B']-
               gradeLowestScore['C']
  ```

**Q2**

# dict initialization & operations

```
>>> gradeLowestScore = {'A':89.5, 'B+':84.5, 'B':79.5,
         'C+':74.5, 'C':69.5, 'D+':64.5, 'D': 59.5, 'F': 0.0}
>>> gradeLowestScore['C']
69.5
>>> gradeLowestScore['C'] = 68.0  # new value for key 'C'
>>> gradeLowestScore.keys()
['A', 'C+', 'C', 'B', 'D+', 'F', 'D', 'B+']
>>> gradeLowestScore.values()
[89.5, 74.5, 68.0, 79.5, 64.5, 0.0, 59.5, 84.5]
>>> gradeLowestScore.items()
[('A', 89.5), ('C+', 74.5), ('C', 68.0), ('B', 79.5), ('D+',
64.5), ('F', 0.0), ('D', 59.5), ('B+', 84.5)]
>>> gradeLowestScore.pop('C')  # remove 'C' item
68.0
>>> 'C' in gradeLowestScore
False
>>> 'D' in gradeLowestScore
True
```

**Q3-4**

## dict's *get* method

- What if we try to find the lowest score for an "E" grade?

-
  ```
  >>> gradeLowestScore['E']
  Traceback (most recent call last):
    File "<pyshell#2>", line 1, in <module>
      gradeLowestScore['E']
  KeyError: 'E'
  ```
- The **get** method has a similar purpose, but lets us provide a value to return if the key we search for is not in the dictionary:

-
  ```
  >>> gradeLowestScore.get('E', 'No such key')
  ```

```
‘No such key'
```

## Two main dictionary uses

- A collection of similar objects
  - Designed for fast lookup by key
- Storing different properties of a single object

## Use 1: Collection of similar objects

- Examples:
  - A movie database in which we use the title as the key and look up the director.
  - A phone database in which we use the person's name as the key and look up the phone number
- In-class exercise
  - Create a concordance for a text file.
  - This is just a list of words in the file and the line numbers on which each word occurs

**Q5-6**

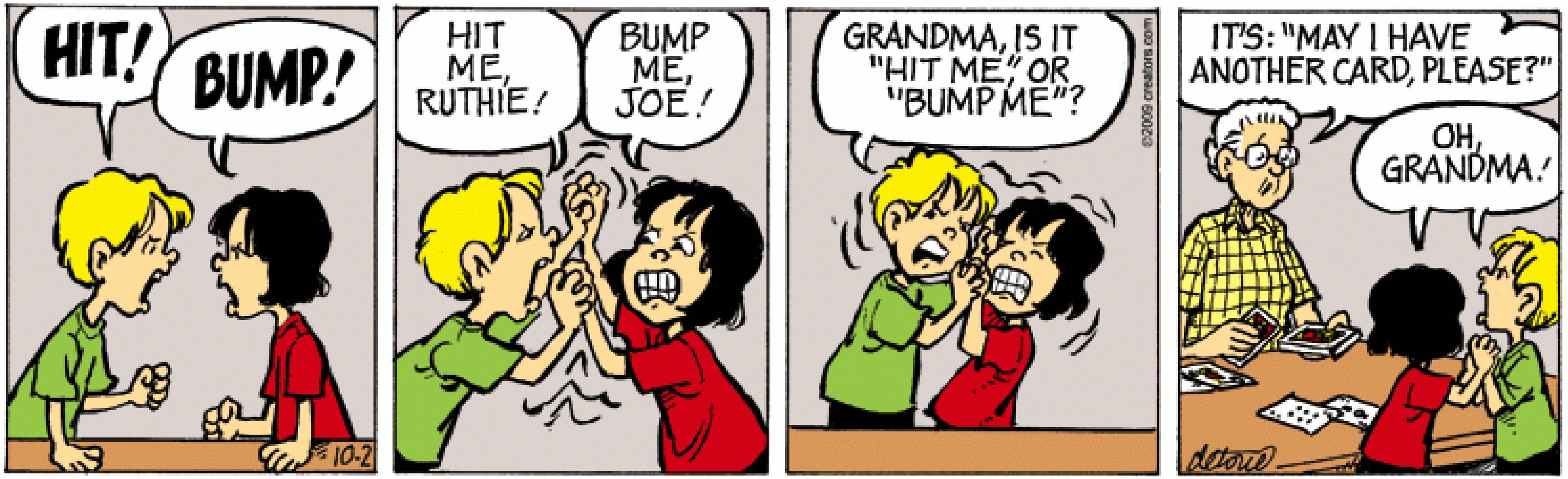
## Towards a movie "database"

- Create a dictionary of movie directors:

```
director = {}
director['Star Wars'] = 'George Lucas'
director['The Godfather'] = 'Francis Ford Coppola'
director['American Graffiti'] = 'George Lucas'
director['Princess Bride'] = 'Rob Reiner'

print director['Star Wars'], 'directed', 'Star Wars.'
```

This example is adapted from *Object-Oriented Programming in Python* by Michael Goldwasser and David Letscher


HIT!
BUMP!
10-2
HIT ME, RUTHIE!
BUMP ME, JOE!
GRANDMA, IS IT "HIT ME," OR "BUMP ME"?
©2009 creators.com
IT'S: "MAY I HAVE ANOTHER CARD, PLEASE?"
OH, GRANDMA!
detorie

## Use 2: Properties of a single object

- Represent a card (blackjack) as a dictionary
- properties: 'cardName', ' suit', ' value'

```
# A card is represented by a dictionary with keys
# cardName, suit, and value

def makeCard (cardName, suit):
    card = {}
    card['suit'] = suit
    card['cardName'] = cardName
    card['value'] = cardValue(cardName)
    return card
```