Dictionaries

Dictionaries

- Our old familiar data types:
  - **int**eger: 1, 13, 42
  - **float**ing point: 3.1415, 0.25, 7502.34
  - **str**ing: “a”, “abc”, “Fred”, “Now is the time\n”
  - **list**: [1, 2, 3], [“Fred”, “John”, “Mary”, “Sue”]
  - **tuple**: (1,2,3), (“Fred”, “John”, “Mary”, “Sue”)
- Another data-type
  - **dictionary**: {'Fred':3, 'Judy':6, 'Elaine':9, 'Mark':2}

Dictionaries

- key : value pairs
  - engGer = {'one': 'eins', 'two': 'zwei', 'three': 'drei'}
- Order is not important
  - Items **not** stored in order
- **Reference values by keys**, not index #, not values!
  - engGer['two'] → “zwei”
  - engGer['four'] → ***KeyError***
  - engGer['eins'] → ***KeyError***

Dictionaries

- Keys must be *immutable* and *unique*
  - number, string, tuple (NOT list)
  - If duplicate keys, **last** one wins

```
numbers = {8: 'eight', 3: 'three', 6: 'six', 9: 'nine', 3: 'drei'}
print (numbers)
```

- Values can be any data-type
  - number, string, list, tuple, …

Dictionary Examples

```
addrBook = {"Fred": ["Flintstone", "123 Foo St", "1234567"],
            "Barney": ["Rubble", "456 Bar Ave", "9876543"],
            "Wilma": ["Granite", "65 Baz Blvd", "6543210"]  }

data = {"544441234”:
            ["Flintstone, Fred", "123 Foo St", "1234567"],
        "123455432”:
            ["Rubble, Barney", "456 Bar Ave", "9876543"],
        "987654321”:
            ["Granite, Wilma", "65 Baz Blvd", "6543210"]  }
```

Working with Dictionaries

- ***in*** tests if **key** is in dictionary (not the value!)
  - e.g., John ***in*** addrBook #True if key “John” is there
  - Can use to guard against KeyError exceptions
  - ***not in*** tests if key is **not** in dictionary

- Get number of items:
  - len(<dictionary name>)

Dictionaries are Mutable

- Can create empty dictionary

```
myDictionary = {}
```

- Can add, delete, change items

```
engGer = {'one': 'eins', 'two': 'zwei', 'three': 'drei'}
engGer['four'] = 'veer'
engGer['four'] = 'vier'

del engGer['four']
```

Dictionary Methods

- **<dictionary name>.<method>()**
- get(key): returns value for key (no KeyError)
- get(key, **alt**): same, but returns **alt** if key absent
- keys(): returns all keys
- values(): returns all values
- items(): returns all key:value pairs
- pop(key): returns value for key; deletes key:value from dictionary
- popitem(): returns *arbitrary (not random)* key:value pair; deletes it from dictionary
- update({k1 : v1, k2:v2}): adds new key:value pairs to dictionary
- clear(): empties the dictionary

Looping through Dictionary

- Using ***for*** loop with dictionary:

for <key> in <dictionary>:

    <do something with key or <dictionary>[key]>

- Example:

```
d = {'one': 'eins', 'two': 'zwei', 'three': 'drei'}
for key in d:
    print('key is: ', key)
    print('value is: ', d[key])
```

Looping through Dictionary

Using ***while*** loop with dictionary. Example:

```
while len(d) > 0:
    item = d.popitem()
    print('Item is: ',item)
    print('Remaining dictionary is: ', d)
```

Remember, popitem() will remove item from dictionary.

card_dealer.py

Using a dictionary to simulate a deck of cards

- see *card_dealer.py* on Canvas

Dealing cards:

```
for count in range(number):
        card, value = deck.popitem()
        print(card)
        hand_value += value
```

What do you notice? See here for more information on arbitrary versus random[11]