

starting out with >>>

# PYTHON®

FIFTH EDITION

## CHAPTER 9

# Dictionaries and Sets



TONY GADDIS

# Topics

- **Dictionaries**
- **Sets**
- **Serializing Objects**

# DNA Count

- **DNA Deoxyribonucleic acid**
  - "The polymer carries genetic instructions for the development, functioning, growth and reproduction of all known organisms and many viruses. "
- **Part of the building blocks of DNA are 4 nitrogen containing nucleobases**
  - cytosine [C], guanine [G], adenine [A] or thymine [T]

# DNA Data

- Massive amounts of work to catalog and decode DNA in organisms has been done.
- <https://www.kaggle.com/datasets/nageshsingh/dna-sequence-dataset?select=dog.txt>
- ATGCCACAGCTAGATACATCCACCTGATTTATTATA  
ATCTTTTCAATATTTCTCACCCCTCTTCATCCTATTTC  
AACTAAAAATTTCAAATCACTACTACCCAGAAAAC  
CCGATAACCAAATCTGCTAAAATTGCTGGTCAACA  
TAATCCTTGAGAAAACAAATGAACGAAAATCTATTC  
GCTTCTTTCGCTGCCCCCTCAATAA

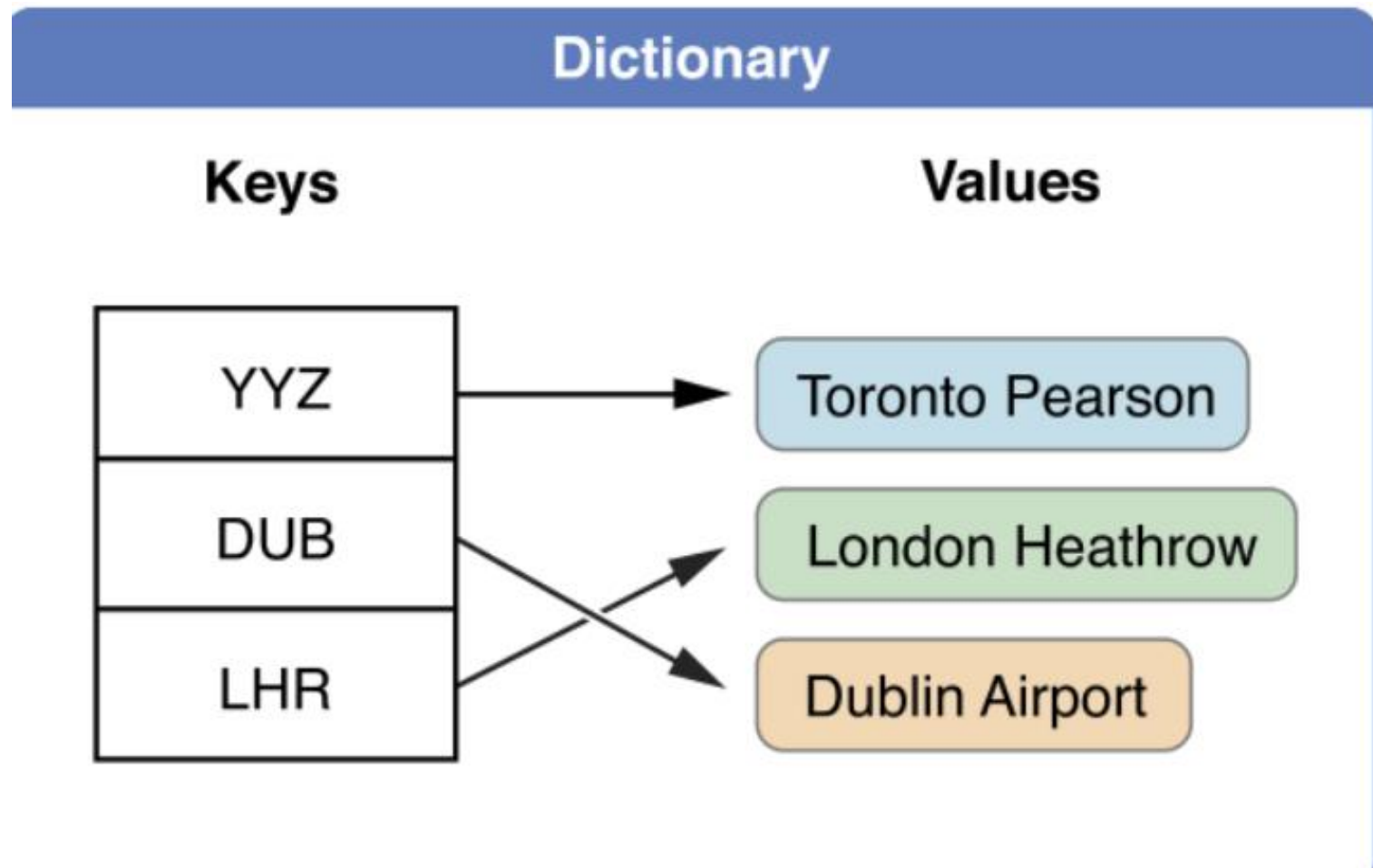
# DNA Counts

- **Write a function that given a string that represents a portion of DNA returns the frequency of the four nucleobases**
- **cytosine [C], guanine [G], adenine [A] or thymine [T]**

# Dictionaries

- **Dictionary**: data structure that stores a collection of *key-value pairs*
  - Each element consists of a *key* and a *value*
    - Often referred to as *mapping* of key to value
    - Key must be an immutable object
    - A real world dictionary, the words are the keys and the definitions are the values
    - Given the word you can find the value ***quickly***
  - To retrieve a specific value, use the key associated with it
  - Format for creating a dictionary with given values  
*dictionary = {key1:val1, key2:val2}*

# Visualization of Dictionary



- <https://docs.swift.org/swift-book/LanguageGuide/CollectionTypes.html>

# Retrieving a Value from a Dictionary

- Prior to Python 3.7 the keys in a dictionary are in no discernible order from the client's perspective
- Python 3.7 and later, dictionaries maintain keys in *insertion order*
- General format for retrieving value from dictionary: *dictionary[key]*
  - If `key` in the dictionary, associated value is returned, otherwise, `KeyError` exception is raised
- Test whether a key is in a dictionary using the `in` and `not in` operators
  - Helps prevent `KeyError` exceptions



# Adding Elements to an Existing Dictionary

- Dictionaries are mutable objects
- To add a new key-value pair:

*dictionary[key] = value*

- If key exists in the dictionary, the value associated with it will be changed
- if the key doesn't exist this adds the *key-value* pair to the dictionary

# Deleting Elements From an Existing Dictionary

- To remove a key-value pair:  
**d.pop(key)**
  - If key is not in the dictionary, `KeyError` exception is raised
  - OR `del dictionary[key]`

# Getting the Number of Elements and Mixing Data Types

- **len function**: used to obtain number of key-value pairs in a dictionary
- **Keys must be immutable objects**, but associated values can be any type of object
  - One dictionary can include keys of several different immutable types. Heterogeneous.
- **Values stored in a single dictionary can be of different types**

# Creating an Empty Dictionary and Using `for` Loop to Iterate Over a Dictionary

- **To create an empty dictionary:**
  - Use `{ }`
  - Use built-in function `dict ( )`
  - Elements can be added to the dictionary as program executes
- **Use a `for` loop to iterate over a dictionary**
  - General format: `for key in dictionary:`

# Some Dictionary Methods

- **clear method:** deletes all the elements in a dictionary, leaving it empty
  - Format: `dictionary.clear()`
- **get method:** gets a value associated with specified key from the dictionary
  - Format: `dictionary.get(key, default)`
    - `default` is returned if `key` is not found
  - Alternative to `[]` operator
    - Cannot raise `KeyError` exception

# Some Dictionary Methods (cont'd.)

- **items method:** returns all the dictionaries keys and associated values
  - Format: `dictionary.items()`
  - Returned as a *dictionary view*
    - Each element in dictionary view is a tuple which contains a key and its associated value
    - Use a `for` loop to iterate over the tuples in the sequence
    - Can use a variable which receives a tuple, or can use two variables which receive key and value

# Some Dictionary Methods (cont'd.)

- **keys method:** returns all the dictionaries keys as a sequence
  - Format: `dictionary.keys()`
- **pop method:** returns value associated with specified key and removes that key-value pair from the dictionary
  - Format: `dictionary.pop(key, default)`
    - `default` is returned if `key` is not found

# Some Dictionary Methods (cont'd.)

- **popitem method**: returns a randomly selected key-value pair and removes that key-value pair from the dictionary
  - Format: `dictionary.popitem()`
  - Key-value pair returned as a tuple
- **values method**: returns all the dictionaries values as a sequence
  - Format: `dictionary.values()`
  - Use a `for` loop to iterate over the values



# Some Dictionary Methods (cont'd.)

**Table 9-1** Some of the dictionary methods

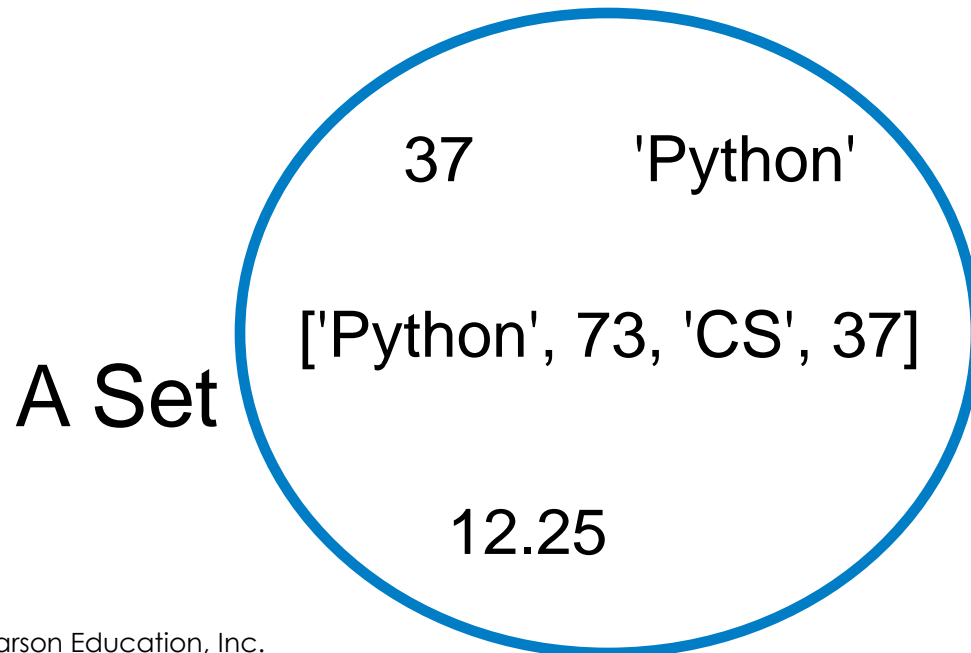
Method	Description
<code>clear</code>	Clears the contents of a dictionary.
<code>get</code>	Gets the value associated with a specified key. If the key is not found, the method does not raise an exception. Instead, it returns a default value.
<code>items</code>	Returns all the keys in a dictionary and their associated values as a sequence of tuples.
<code>keys</code>	Returns all the keys in a dictionary as a sequence of tuples.
<code>pop</code>	Returns the value associated with a specified key and removes that key-value pair from the dictionary. If the key is not found, the method returns a default value.
<code>popitem</code>	Returns a randomly selected key-value pair as a tuple from the dictionary and removes that key-value pair from the dictionary.
<code>values</code>	Returns all the values in the dictionary as a sequence of tuples.

# Dictionary Example

- **Use a dictionary to determine which "word" occurs the most in a text.**
- **What will be the keys?**
- **What will be the values?**

# Sets

- **Set: object that stores a collection of data in same way as mathematical set**
  - Items are unique, duplicates don't exist in a set
  - Set is unordered, from the client's perspective
  - Elements can be of different data types



# Creating a Set

- **set function**: used to create a set
  - Simple set creation
    - `set1 = {12, 'Python', 37, 73}`
  - For empty set, call `set()`
  - For non-empty set, call `set(argument)` where *argument* is an object that contains iterable elements
    - e.g., *argument* can be a list, string, or tuple
    - If *argument* is a string, each character becomes a set element
      - For set of strings, pass them to the function as a list
    - If *argument* contains duplicates, only one of the duplicates will appear in the set

# Creating Data Types

- **List:**

- `data = [7, 37, 5, 37, 12, 37.5]`

- **List of lists:**

- `table = [[1, 2], [3, 7], [19, 73]]`

- **String:**

- `name = 'Python Language'`

- **Tuple:**

- `tup1 = (37, 'Python', 73, 12, 12)`

- **Dictionary:**

- `freq_map = {'Python': 3, 'Java': 7}`

- **Set:**

- `lang_set= {'Python', 'Java', 'C++'}`

21



# Sets are Unordered

- Unlike the keys of a dictionary (which are a set, no duplicates), the elements in a Python set are unordered from the client's perspective.

```
>>> lang_set= {'Python', 'Java', 'C++',  
...           12, 'Swift', 37, 12}  
>>> lang_set  
{'Java', 'Python', 37, 'C++', 'Swift', 12}
```

# Getting the Number of and Adding Elements

- **len function**: returns the number of elements in the set
- **Sets are mutable objects**
- **add method**: adds an element to a set
  - What if set already contains that element?
- **update method**: adds a group of elements to a set
  - Argument must be a sequence containing iterable elements, and each of the elements is added to the set

# Deleting Elements From a Set

- **remove and discard methods:** remove the specified item from the set
  - The item that should be removed is passed to both methods as an argument
  - Behave differently when the specified item is not found in the set
    - `remove` method raises a `KeyError` exception
    - `discard` method does not raise an exception
- **clear method:** clears all the elements of the set

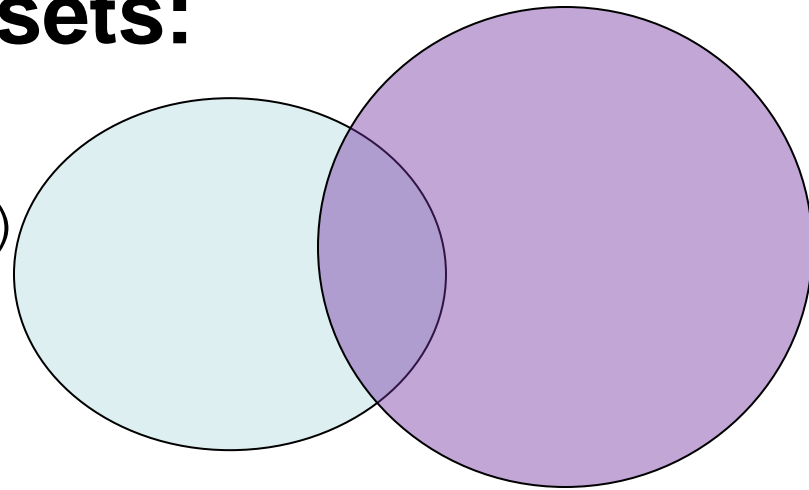


# Using the `for` Loop, `in`, and `not in` Operators With a Set

- A `for` loop can be used to iterate over elements in a set
  - General format: `for item in set:`
  - The loop iterates once for each element in the set
- The `in` operator can be used to test whether a value exists in a set
  - Similarly, the `not in` operator can be used to test whether a value does not exist in a set

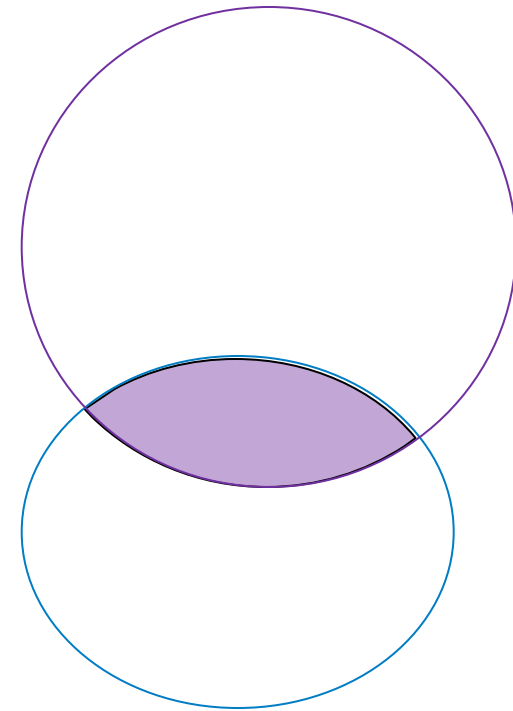
# Finding the Union of Sets

- **Union of two sets**: a set that contains all the elements of both sets
- **To find the union of two sets:**
  - Use the `union` method
    - Format: `set1.union(set2)`
  - Use the `|` operator
    - Format: `set1 | set2`
  - Both techniques return a new set which contains the union of both sets



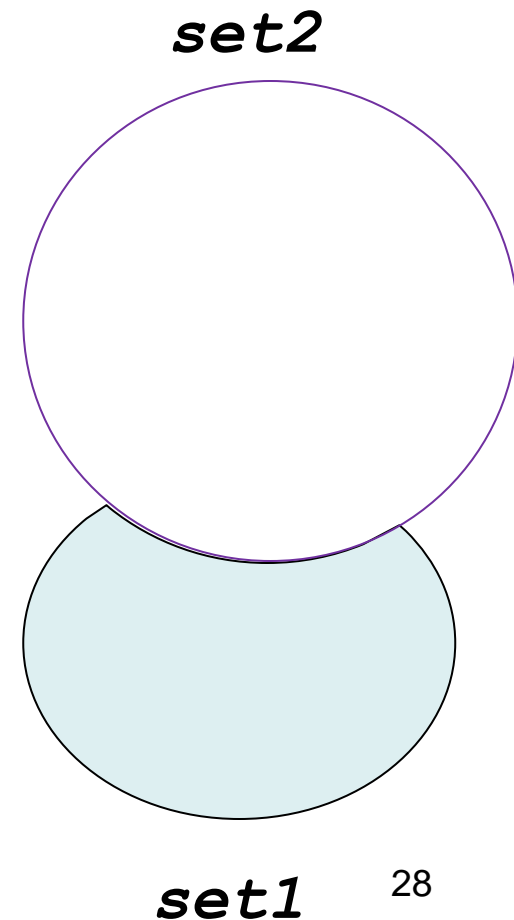
# Finding the Intersection of Sets

- **Intersection of two sets**: a set that contains only the elements found in both sets
- **To find the intersection of two sets:**
  - Use the `intersection` method
    - Format: `set1.intersection(set2)`
  - Use the `&` operator
    - Format: `set1 & set2`
  - Both techniques return a new set which contains the intersection of both sets



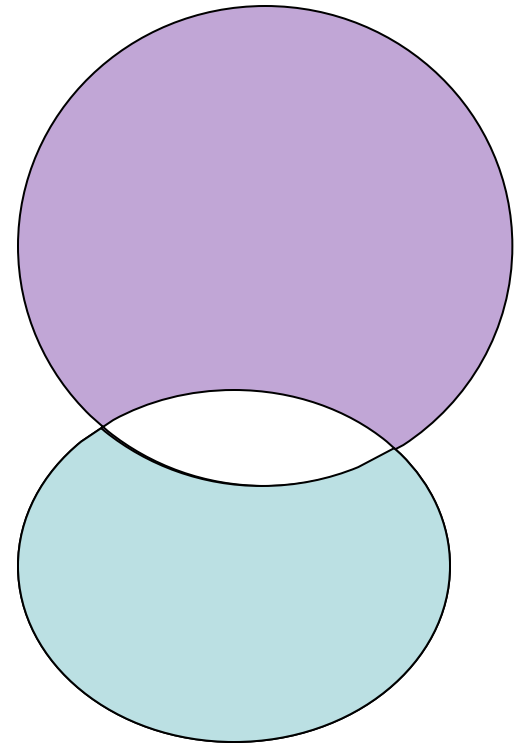
# Finding the Difference of Sets

- **Difference of two sets**: a set that contains the elements that appear in the first set but do not appear in the second set
- **To find the difference of two sets:**
  - Use the `difference` method
    - Format: `set1.difference(set2)`
  - Use the `-` operator
    - Format: `set1 - set2`



# Finding the Symmetric Difference of Sets

- **Symmetric difference of two sets**: a set that contains the elements that are not shared by the two sets
- **To find the symmetric difference of two sets:**
  - Use the `symmetric_difference` method
    - Format:  
`set1.symmetric_difference(set2)`
  - Use the `^` operator
    - Format: `set1 ^ set2`



# Finding Subsets and Supersets

- **Set A is subset of set B if all the elements in set A are included in set B**
- **To determine whether set A is subset of set B**
  - Use the `issubset` method
    - Format: `setA.issubset(setB)`
  - Use the `<=` operator
    - Format: `setA <= setB`

# Finding Subsets and Supersets (cont'd.)

- **Set A is superset of set B if it contains all the elements of set B**
- **To determine whether set A is superset of set B**
  - Use the `issuperset` method
    - Format: `setA.issuperset(setB)`
  - Use the `>=` operator
    - Format: `setA >= setB`

# Serializing Objects

- **Serialize an object**: convert the object to a stream of bytes that can easily be stored in a file
- **Pickling**: serializing an object



# Serializing Objects (cont'd.)

- **To pickle an object:**
  - Import the `pickle` module
  - Open a file for binary writing, 'wb' option
  - Call the `pickle.dump` function
    - Format: `pickle.dump(object, file)`
  - Close the file
- **You can pickle multiple objects to one file prior to closing the file**

# Serializing Objects (cont'd.)

- **Unpickling**: retrieving pickled object
- **To unpickle an object:**
  - Import the `pickle` module
  - Open a file for binary writing, 'rb'
  - Call the `pickle.load` function
    - Format: `pickle.load(file)`
  - Close the file
- **You can unpickle multiple objects from the file**