Introduction

Module Learning Outcomes

After successful completion of this module, you will be able to ...

1. Create and use dictionaries.
2. Create and use sets.

Key questions:

* How is a dictionary different from a list?
* How is a set different from a list?
* How is a set different from a dictionary?

Explorations

Use the pages within this module to explore the following concepts:

* Exploration: [Dictionaries](https://canvas.oregonstate.edu/courses/1928696/pages/exploration-dictionaries) (CLO 1c, MLO 1)
* Exploration: [Sets](https://canvas.oregonstate.edu/courses/1928696/pages/exploration-sets) (CLO 1c, MLO 2)
* Video demo: [dictionaries, sets](https://canvas.oregonstate.edu/courses/1928696/pages/video-demo-dictionaries-sets) (CLO 1c, MLO 1-2)
* [Module 8 Exercise Solutions](https://canvas.oregonstate.edu/courses/1928696/pages/module-8-exercise-solutions)

Optional Resources

* [*Think Python* Chapter 11Links to an external site.](http://greenteapress.com/thinkpython2/html/thinkpython2012.html)
* [*Think Python* Chapter 19, sections 5 - 7Links to an external site.](http://greenteapress.com/thinkpython2/html/thinkpython2020.html#sec227)

Task List

Complete the following assignments and other tasks:

* Sign up for one of the [Assignment 7 - Group Part](https://canvas.oregonstate.edu/courses/1928696/assignments/9073292) groups and do the group assignment  (CLOs 1c, 4, MLOs 1-2).
* Read the Exploration pages and do the interactive exercises on those pages (CLO 1c, MLOs 1-2).
* Do [Assignment 8](https://canvas.oregonstate.edu/courses/1928696/assignments/9073293), which gives you practice using dictionaries and sets (CLO 1c, MLOs 1-2).
* Take [Quiz 8](https://canvas.oregonstate.edu/courses/1928696/quizzes/2825535) (CLO 1c, MLOs 1-2).

# **Exploration: Dictionaries**

## Dictionary Data Types

Another data type that's built-in to Python is called a dictionary. A dictionary is a way of storing key-value pairs. For example:

The first column has the keys and the second column has their associated values (I added extra spaces to align the columns and colons just for ease of reading - it's not required). Each key can only have one value, but the same value can be used for multiple keys. Dictionaries, like lists, are **mutable**. However, unlike lists, you cannot index into a dictionary by the position of an element. You still use the square bracket notation, but instead of a positional index, you give the key that you want the associated value of.

> state\_capitals["Idaho"]  
> state\_capitals["Washington"]

It's easy to add a new key-value pair:

> state\_capitals["Florida"] = "Orlando"  
> state\_capitals["Florida"]

Or to change an existing key-value pair:

> state\_capitals["Florida"] = "Tallahassee"  
> state\_capitals["Florida"]

You can delete an entry using **del**:

> del state\_capitals["Florida"]  
> state\_capitals["Florida"]

If you print a dictionary, it will list all of its key-value pairs:

> print(state\_capitals)

The keys in a dictionary must be immutable, but the associated values can be of any type. The keys are not required to all be the same type. You can even nest dictionaries. Here's an example of a dictionary with some string values and an integer value.

You can declare an empty dictionary like so:

empty\_dict = {}

You can use in and not in to test whether a key is in a dictionary:

> "classification" in capybara  
> "genus" in capybara

You can use len to get the number of key-value pairs in a dictionary:

> len(capybara)

You can use a for loop to iterate through the keys of a dictionary.

for attribute in capybara:  
 print(attribute)

Through Python 3.6, dictionaries were not guaranteed to preserve a particular ordering. As of Python 3.7, dictionaries are guaranteed to preserve insertion order.

The clear method makes a dictionary empty:

> capybara.clear()  
> len(capybara)

As with lists, there are other operators, functions and methods available for working with dictionaries.

## Exercises

(See the module overview for a link to example solutions.)

1. Make a dictionary called "eng\_to\_span" where the keys are the English words "one" through "ten", and the corresponding values are their Spanish translations.

Sample input: NA  
Expected output: NA

2. Using the dictionary from #1, write a loop that prints out both the key and value of each key-value pair, for example the first iteration of the loop should print 'one' 'uno'.

Sample input: NA  
Expected output:   
 'one', 'uno'  
 'two', 'dos'  
 'three', 'tres'  
 'four', 'quatro'  
 'five', 'cinco'  
 'six', 'seis'  
 'seven', 'siete'  
 'eight, 'ocho'  
 'nine', 'nueve'  
 'ten', 'diez'

3. Write a function named some\_squares that takes a positive integer parameter and returns a dictionary where the keys are the integers from 1 through the value of the parameter, and the corresponding values are the squares of those integers.

Sample input: 3  
Expected output: {1: 1, 2: 4, 3: 9}

# **Exploration: Sets**

## Introduction

You're probably familiar with the concept of sets from math classes. A **set** is an unordered collection of elements where every element is unique. Adding a duplicate element doesn't change a set. Non-duplicate elements can be added to a set - sets are **mutable**. However, the elements in a set must be **immutable**, so a set cannot contain lists or dictionaries or sets. Sets can contain values of different types, or be empty.

some\_elements contains an int, a string, a float, a bool, and a tuple - all of which are immutable types.

You can create a set containing 1,2, and 3 in two ways. The first is to use the curly brace notation. The second is to use the set() function to convert a tuple or list with those elements into a set. The argument for the set function must be an iterable (for example a list or tuple), or for an empty set, no argument - just empty parentheses.

my\_new\_set = {1,2,3}  
my\_new\_set = set((1,2,3)) # This passes a tuple, but could pass a different iterable type, such as a list

However, to create a new empty set, you must use the second syntax, since Python will interpret an empty pair of curly braces as an empty dictionary, not an empty set.

You cannot index or slice sets.

You can use in and not in to check whether a value is in the set:

> -7 in some\_ints  
> -7 not in some\_ints

You can add elements to a set or remove elements from a set:

> some\_ints.remove(1)  
> some\_ints  
> some\_ints.add(1)  
> some\_ints

If you add an element that is already in the set, nothing happens.

The len function tells you how many elements are in a set:

> len(some\_elements)

The clear method empties a set:

> some\_elements.clear()  
> len(some\_elements)

### **Set Operations**

Python has operators for performing various set operations. The most common set operations are union, intersection, difference, and symmetric difference. Each of these operations produces a new set and leaves the original sets unchanged.

The intersection of two sets is a new set that contains every element that is in **both** sets.

> set\_A & set\_B

The union of two sets is a new set that contains every element that is in **either** set.

> set\_A | set\_B # the union operator is the vertical bar symbol

The difference of two sets is a new set that contains every element that is in the first set **except for** the elements that are also in the second set. This operation is not symmetric.

> set\_A - set\_B  
> set\_B - set\_A

The symmetric difference of two sets is a new set that contains every element that is in either set except the elements that are in both sets. This is the same as the union minus the intersection.

> set\_A ^ set\_B  
> (set\_A | set\_B) - (set\_A & set\_B)

We can iterate through a set with a for loop:

for el in set\_B:  
 print(el \* 2)

### **Lists vs. sets**

How do you know when to use a list and when to use a set?

1. A set cannot contain duplicate elements, but a list can.
2. A set can only contain values that are immutable, but lists can contain mutable values.
3. A list is ordered - you can know that its elements will be traversed or printed out in a particular order. The same is not true of sets (or dictionaries), which are unordered.
4. Checking whether a certain value is in a list takes time proportional to the length of the list, but doing so in a set is very fast, regardless of the size of the set.

### **Set comprehensions**

A set comprehension looks just like a list comprehension except it uses curly braces instead of square brackets.  For example, if you have a list of names that possibly contains duplicates, and you would like to filter out any names that aren't palindromes (ignoring case) and get rid of duplicates at the same time, you could create a set like so:

{name for name in li if name.lower() == name.lower()[::-1]}

 Checking for equality to the reverse slice checks for palindromes. Using lower() takes care of ignoring case.  The curly braces mean it creates a set, which will automatically exclude any duplicates.

## Exercises

(See the module overview for a link to example solutions.)

1. Write your own function called "unionize" for finding the union of two sets. Your function should take as parameters two sets, and return a new set that is the union of those two sets. Do not use Python's built-in union functionality.

Sample input: {4, 3, 100}, {1, 100}  
Expected output: {4, 3, 1, 100}

2. Write your own function called "intersect" for finding the intersection of two sets. Your function should take as parameters two sets, and return a new set that is the intersection of those two sets. Do not use Python's built-in intersection functionality.

Sample input: {4, 3, 100}, {1, 100}  
Expected output: {100}

3. Write your own function called "sym\_diff" for finding the symmetric difference of two sets.  Your function should take as parameters two sets, and return a new set that is the symmetric difference of those two sets.  Do not use Python's built-in symmetric difference functionality.

Sample input: {4, 3, 100}, {1, 100}  
Expected output: {4, 3, 1}

# **Review - Dictionaries & Sets**

## Key Take-Aways

At this point, you should be able to answer all of the following questions.

* What is a dictionary? How is it different from a list?
* In what ways can you modify a dictionary?
* What are some other operations/functions/methods for dictionaries?
* What is a set? How is it different from a list?
* What are some of the operations/functions/methods for sets?

You should now be able to create and use dictionaries, as well as create and use sets.

If you are unsure of any of these answers, take some time to review the course materials. Feel free to post questions in Ed Discussion or Teams. Next week's lessons will be easier if you have invested in mastering this week's learning outcomes.

Check the **Explorations** and **Task List** in the **Lesson 8 - Overview** to review learning materials and make sure you have completed all required activities for this week.