UNIVERSIDAD DE GUADALAJARA

CUCEI

DIVISIÓN DE ELECTRÓNICA Y COMPUTACIÓN

DEPARTAMENTO DE CIENCIAS COMPUTACIONALES

TAREA No. 5

TEMA: DICCIONARIOS Y CONJUNTOS

ARELLANO GRANADOS ANGEL MARIANO

FUNDAMENTOS FILOSOFICOS DE LA COMPUTACION

D13 2021-A

LUIS FELIPE MARISCAL LUGO

**MARCO TEÓRICO:**

**APARTADO 10.1**

Dictionaries

# A dictionary is an object that stores a collection of data. Each element in a dictionary has two parts: a key and a value. You use a key to locate a specific value.

In Python, a dictionary is an object that stores a collection of data. Each element that is stored in a dictionary has two parts: a key and a value. In fact, dictionary elements are commonly referred to as key-value pairs. When you want to retrieve a specific value from a dictionary, you use the key that is associated with that value. This is similar to the process of looking up a word in the Merriam-Webster dictionary, where the words are keys and the definitions are values.

# Creating a Dictionary

You can create a dictionary by enclosing the elements inside a set of curly braces ( {} ). An element consists of a key, followed by a colon, followed by a value. The elements are separated by commas. The following statement shows an example:

phonebook = {'Chris':'555-1111', 'Katie':'555-2222', 'Joanne':'555-3333'}

In this example the keys and the values are strings. The values in a dictionary can be objects of any type, but the keys must be immutable objects. For example, keys can be strings, integers, floating-point values, or tuples. Keys cannot be lists or any other type of immutable object.

# Retrieving a Value from a Dictionary

The elements in a dictionary are not stored in any particular order. For example, look at the following interactive session in which a dictionary is created and its elements are displayed:

>>> phonebook = {'Chris':'555-1111', 'Katie':'555-2222', 'Joanne':'555-3333'}

>>> phonebook

{'Chris': '555-1111', 'Joanne': '555-3333', 'Katie': '555-2222'}

Notice that the order in which the elements are displayed is different than the order in which they were created. This illustrates how dictionaries are not sequences, like lists, tuples, and strings. As a result, you cannot use a numeric index to retrieve a value by its position from a dictionary. Instead, you use a key to retrieve a value.

To retrieve a value from a dictionary, you simply write an expression in the following general format:

dictionary\_name[key]

In the general format, dictionary\_name is the variable that references the dictionary, and key is a key. If the key exists in the dictionary, the expression returns the value that is associatedwith the key. If the key does not exist, a KeyError exception is raised. The following interactive session demonstrates:

1 >>> phonebook = {'Chris':'555-1111', 'Katie':'555-2222',

'Joanne':'555-3333'}

2 >>> phonebook['Chris']

3 '555-1111'

4 >>> phonebook['Joanne']

5 '555-3333'

6 >>> phonebook['Katie']

7 '555-2222'

8 >>> phonebook['Kathryn']

Traceback (most recent call last):

File "<pyshell#5>", line 1, in <module>

phonebook['Kathryn']

KeyError: 'Kathryn'

# Using the in and not in Operators to Test for a Value in a Dictionary

As previously demonstrated, a KeyError exception is raised if you try to retrieve a value from a dictionary using a nonexistent key. To prevent such an exception, you can use the in operator to determine whether a key exists before you try to use it to retrieve a value.

The following interactive session demonstrates:

1 >>> phonebook = {'Chris':'555-1111', 'Katie':'555-2222',

'Joanne':'555-3333'} e

2 >>> if 'Chris' in phonebook: e

3 print(phonebook['Chris']) e e

4

5 555-1111

The if statement in line 2 determines whether the key 'Chris' is in the phonebook dictionary. If it is, the statement in line 3 displays the value that is associated with that key.

You can also use the not in operator to determine whether a key does not exist, as demonstrated in the following session:

1 >>> phonebook = {'Chris':'555-1111', 'Katie':'555-2222'} e

2 >>> if 'Joanne' not in phonebook: e

3 print('Joanne is not found.') e e

4

5 Joanne is not found.

# Adding Elements to an Existing Dictionary

Dictionaries are mutable objects. You can add new key-value pairs to a dictionary with an assignment statement in the following general format:

dictionary\_name[key] = value

In the general format, dictionary\_name is the variable that references the dictionary, and key is a key. If key already exists in the dictionary, its associated value will be changed to value. If the key does not exist, it will be added to the dictionary, along with value as its associated value. The following interactive session demonstrates:

1 >>> phonebook = {'Chris':'555-1111', 'Katie':'555-2222',

'Joanne':'555-3333'} e

2 >>> phonebook['Joe'] = '555-0123' e

3 >>> phonebook['Chris'] = '555-4444' e

4 >>> phonebook e

5 {'Chris': '555-4444', 'Joanne': '555-3333', 'Joe': '555-0123',

'Katie': '555-2222'}

# Deleting Elements

You can delete an existing key-value pair from a dictionary with the del statement. Here is the general format:

del dictionary\_name[key]

In the general format, dictionary\_name is the variable that references the dictionary, and key is a key. After the statement executes, the key and its associated value will be deleted from the dictionary. If the key does not exist, a KeyError exception is raised.

# Getting the Number of Elements in a Dictionary

the keys in a dictionary must be immutable objects, but their associated values can be any type of object. For example, the values can be lists, as demonstratedm in the following interactive session. In this session we create a dictionary in which the keys are student names and the values are lists of test scores.

1 >>> test\_scores = { 'Kayla' : [88, 92, 100],

2 'Luis' : [95, 74, 81],

3 'Sophie' : [72, 88, 91],

4 'Ethan' : [70, 75, 78] }

5 >>> test\_scores

6 {'Kayla': [88, 92, 100], 'Sophie': [72, 88, 91], 'Ethan': [70, 75, 78],

7 'Luis': [95, 74, 81]}

8 >>> test\_scores['Sophie']

9 [72, 88, 91]

10 >>> kayla\_scores = test\_scores['Kayla']

11 >>> print(kayla\_scores)

12 [88, 92, 100]

The values that are stored in a single dictionary can be of different types. For example, one element’s value might be a string, another element’s value might be a list, and yet another element’s value might be an integer. The keys can be of different types, too, as long as they are immutable. The following interactive session demonstrates how different types can be mixed in a dictionary:

1 >>> mixed\_up = {'abc':1, 999:'yada yada', (3, 6, 9):[3, 6, 9]}

2 >>> mixed\_up

3 {(3, 6, 9): [3, 6, 9], 'abc': 1, 999: 'yada yada'}

The following interactive session gives a more practical example. It creates a dictionary that contains various pieces of data about an employee:

1 >>> employee = {'name' : 'Kevin Smith', 'id' : 12345, 'payrate' : 25.75 }

2 >>> employee

3 {'payrate': 25.75, 'name': 'Kevin Smith', 'id': 12345}

# Creating an Empty Dictionary

Sometimes you need to create an empty dictionary and then add elements to it as the program executes. You can use an empty set of curly braces to create an empty dictionary, as demonstrated in the following interactive session:

1 >>> phonebook = {}

2 >>> phonebook['Chris'] = '555-1111'

3 >>> phonebook['Katie'] = '555-2222'

4 >>> phonebook['Joanne'] = '555-3333'

5 >>> phonebook e

6 {'Chris': '555-1111', 'Joanne': '555-3333', 'Katie': '555-2222'}

You can also use the built-in dict() method to create an empty dictionary, as shown in thefollowing statement:

phonebook = dict()

After this statement executes, the phonebook variable will reference an empty dictionary.

# Using the for Loop to Iterate over a Dictionary

You can use the for loop in the following general format to iterate over all the keys in a dictionary:

for var in dictionary:

statement

statement

etc.

In the general format, var is the name of a variable and dictionary is the name of a dictionary. This loop iterates once for each element in the dictionary. Each time the loop iterates, var is assigned a key. The following interactive session demonstrates:

# Some Dictionary Methods

|  |  |
| --- | --- |
| Method | Description |
| clear | Clears the contents of a dictionary. |
| get | 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. |
| items | Returns all the keys in a dictionary and their associated values as  a sequence of tuples. |
| keys | Returns all the keys in a dictionary as a sequence of tuples. |
| pop | 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. |
| popitem | Returns a randomly selected key-value pair as a tuple from the  dictionary and removes that key-value pair from the dictionary. |
| values | Returns all the values in the dictionary as a sequence of tuples. |

**APARTADO 10.2**

# Sets

A set is an object that stores a collection of data in the same way as mathematical sets. Here are some important things to know about sets:

* All the elements in a set must be unique. No two elements can have the same value.
* Sets are unordered, which means that the elements in a set are not stored in any particular order.
* The elements that are stored in a set can be of different data types.

# Creating a Set

To create a set, you have to call the built-in set function. Here is an example of how you create an empty set:

myset = set()

After this statement executes, the myset variable will reference an empty set. You can also pass one argument to the set function. The argument that you pass must be an object that contains iterable elements, such as a list, a tuple, or a string. The individual elements of the object that you pass as an argument become elements of the set. Here is an example:

myset = set(['a', 'b', 'c'])

# Getting the Number of Elements in a Set

As with lists, tuples, and dictionaries, you can use the len function to get the number of elements in a set. The following interactive session demonstrates:

1 >>> myset = set([1, 2, 3, 4, 5]) e

2 >>> len(myset) e

3 5

# Adding and Removing Elements

Sets are mutable objects, so you can add items to them and remove items from them. You use the add method to add an element to a set. The following interactive session demonstrates:

1 >>> myset = set()

2 >>> myset.add(1)

3 >>> myset.add(2)

4 >>> myset.add(3)

5 >>> myset

6 {1, 2, 3}

7 >>> myset e.add(2)

8 >>> myset

9 {1, 2, 3}

You can remove an item from a set with either the remove method or the discard method. You pass the item that you want to remove as an argument to either method, and that item is removed from the set. The only difference between the two methods is how they behave when the specified item is not found in the set. The remove method raises a KeyError exception, but the discard method does not raise an exception.

You can clear all the elements of a set by calling the clear method.

# Using the for Loop to Iterate over a Set

You can use the for loop in the following general format to iterate over all the elements in a set:

for var in set:

statement

statement

etc.

In the general format, var is the name of a variable and set is the name of a set. This loop iterates once for each element in the set. Each time the loop iterates, var is assigned an element.

# Using the in and not in Operators to Test for a Value in a Set

You can use the in operator to determine whether a value exists in a set. The following interactive session demonstrates:

1 >>> myset = set([1, 2, 3])

2 >>> if 1 in myset:

3 print('The value 1 is in the set.')

4 The value 1 is in the set.

You can also use the not in operator to determine if a value does not exist in a set, as demonstrated in the following session:

1 >>> myset = set([1, 2, 3])

2 >>> if 99 not in myset:

3 print('The value 99 is not in the set.')

4 The value 99 is not in the set.

# Finding the Union of Sets

The union of two sets is a set that contains all the elements of both sets. In Python, you can call the union method to get the union of two sets. Here is the general format:

set1.union(set2)

In the general format, set1 and set2 are sets. The method returns a set that contains the elements of both set1 and set2. The following interactive session demonstrates:

1 >>> set1 = set([1, 2, 3, 4])

2 >>> set2 = set([3, 4, 5, 6])

3 >>> set3 = set1.union(set2)

4 >>> set3

5 {1, 2, 3, 4, 5, 6}

# Finding the Intersection of Sets

The intersection of two sets is a set that contains only the elements that are found in both sets. In Python, you can call the intersection method to get the intersection of two sets. Here is the general format:

set1.intersection(set2)

In the general format, set1 and set2 are sets. The method returns a set that contains the elements that are found in both set1 and set2. The following interactive session demonstrates:

1 >>> set1 = set([1, 2, 3, 4])

2 >>> set2 = set([3, 4, 5, 6])

3 >>> set3 = set1.intersection(set2)

4 >>> set3

5 {3, 4}

# Finding the Difference of Sets

The difference of set1 and set2 are the elements that appear in set1 but do not appear in set2. In Python, you can call the difference method to get the difference of two sets. Here is the general format:

set1.difference(set2)

In the general format, set1 and set2 are sets. The method returns a set that contains the elements that are found in set1 but not in set2. The following interactive session demonstrates:

1 >>> set1 = set([1, 2, 3, 4])

2 >>> set2 = set([3, 4, 5, 6])

3 >>> set3 = set1.difference(set2)

4 >>> set3

5 {1, 2}

# Finding the Symmetric Difference of Sets

The symmetric difference of two sets is a set that contains the elements that are not shared by the sets. In other words, it is the elements that are in one set but not in both. In Python, you can call the symmetric\_difference method to get the symmetric difference of two sets. Here is the general format:

set1.symmetric\_difference(set2)

In the general format, set1 and set2 are sets. The method returns a set that contains the elements that are found in either set1 or set2 but not both sets. The following interactive session demonstrates:

1 >>> set1 = set([1, 2, 3, 4])

2 >>> set2 = set([3, 4, 5, 6])

3 >>> set3 = set1.symmetric\_difference(set2)

4 >>> set3

5 {1, 2, 5, 6}

# Finding Subsets and Supersets

Suppose you have two sets and one of those sets contains all of the elements of the other set. Here is an example:

set1 = set([1, 2, 3, 4])

set2 = set([2, 3])

In this example, set1 contains all the elements of set2, which means that set2 is a subset of set1. It also means that set1 is a superset of set2. In Python, you can call the issubset method to determine whether one set is a subset of another. Here is the general format:

set2.issubset(set1)

In the general format, set1 and set2 are sets. The method returns True if set2 is a subset of set1. Otherwise, it returns False. You can call the issuperset method to determine whether one set is a superset of another. Here is the general format:

set1.issuperset(set2)

In the general format, set1 and set2 are sets. The method returns True if set1 is a superset of set2. Otherwise, it returns False. The following interactive session demonstrates:

1 >>> set1 = set([1, 2, 3, 4])

2 >>> set2 = set([2, 3])

3 >>> set2.issubset(set1)

4 True

5 >>> set1.issuperset(set2)

6 True

# REFERENCIAS:

* Gaddis, T. (2012). starting out with python. Google. <https://www.google.com/url?sa=t&rct=j&q=&esrc=s&source=web&cd=&cad=rja&uact=8&ved=2ahUKEwi_rITUy6bvAhUDWqwKHXywCo0QFjAAegQIARAD&url=http%3A%2F%2Findex-of.es%2FPython%2FStarting%2520Out%2520With%2520%2520Python%2520Second%2520Edition.pdf&usg=AOvVaw3s1kmo1BGL5EkeL2ELSN9N>

**CONCLUSIÓN:**

Los diccionarios y conjuntos son herramientas muy útiles que podremos usar para almacenar, manejar y usar muchos datos dentro de nuestros programas, ya que sino organizamos nuestros datos en un programa este puede ser realmente difícil de comprender, arrojando errores de lugares que no esperamos, por ello tenerlos en un lugar mas organizado nos ayudara como programadores en un futuro cercano.