UNIVERSIDAD DE GUADALAJARA

CUCEI

DIVISIÓN DE ELECTRÓNICA Y COMPUTACIÓN

DEPARTAMENTO DE CIENCIAS COMPUTACIONALES

TAREA No. 4

TEMA: LISTAS

ARELLANO GRANADOS ANGEL MARIANO

FUNDAMENTOS FILOSOFICOS DE LA COMPUTACION

D13 2021-A

LUIS FELIPE MARISCAL LUGO

**MARCO TEÓRICO:**

Sequences

# A sequence is an object that holds multiple items of data, stored one after the other. You can perform operations on a sequence to examine and manipulate the items stored in it.

Sequence types: lists and tuples.

Both lists and tuples are sequences that can hold various types of data. The difference between lists and tuples is simple: a list is mutable, which means that a program can change its contents, but a tuple is immutable, which means that once it is created, its contents cannot be changed.

Introduction to Lists

A A list is an object that contains multiple data items. Lists are mutable, which means that their contents can be changed during a program’s execution.

A list is an object that contains multiple data items. Each item that is stored in a list is called an element. Here is a statement that creates a list of integers:

even\_numbers = [2, 4, 6, 8, 10]

A list can hold items of different types, as shown in the following example:

info = ['Alicia', 27, 1550.87]

You can use the print function to display an entire list, as shown here:

numbers = [5, 10, 15, 20]

print(numbers)

In this example, the print function will display the elements of the list like this:

[5, 10, 15, 20]

Python also has a built-in list() function that can convert certain types of objects to lists.

You can use a statement such as the following to convert the range function’s iterable object to a list:

numbers = list(range(5))

# The Repetition Operator

\* symbol is a sequence (such as a list) and the operand on the right side is an integer, it becomes the repetition operator.

In the general format, list is a list and n is the number of copies to make. The following

interactive session demonstrates:

numbers = [0] \* 5

print(numbers)

[0, 0, 0, 0, 0]

Here is another interactive mode demonstration:

numbers = [1, 2, 3] \* 3 e

print(numbers) e

[1, 2, 3, 1, 2, 3, 1, 2, 3]

# Iterating over a List with the for Loop

You can iterate over a list with the for loop, as shown here:

numbers = [99, 100, 101, 102]

for n in numbers:

print(n)

If we run this code, it will print:

99

100

101

102

# Indexing

you can access the individual elements in a list is with an index. Each element in a list has an index that specifies its position in the list. Indexing starts at 0, so the index of the first element is 0, the index of the second element is 1, and so forth.

For example, the following statement creates a list with 4 elements:

my\_list = [10, 20, 30, 40]

The indexes of the elements in this list are 0, 1, 2, and 3. We can print the elements of the list with the following statement:

print(my\_list[0], my\_list[1], my\_list[2], my\_list[3])

The following loop also prints the elements of the list:

index = 0

while index < 4:

print(my\_list[index])

index += 1

You can also use negative indexes with lists, to identify element positions relative to the end of the list. The Python interpreter adds negative indexes to the length of the list to determine the element position.

my\_list = [10, 20, 30, 40]

print(my\_list[-1], my\_list[-2], my\_list[-3], my\_list[-4])

In this example, the print function will display:

40 30 20 10

An IndexError exception will be raised if you use an invalid index with a list. For example, look at the following code:

# This code will cause an IndexError exception.

my\_list = [10, 20, 30, 40]

index = 0

while index < 5:

print(my\_list[index])

index += 1

The last time that this loop iterates, the index variable will be assigned the value 5, which is an invalid index for the list. As a result, the statement that calls the print function Will cause an IndexError exception to be raised.

# The len Function

Python has a built-in function named len that returns the length of a sequence, such as a list. The following code demonstrates:

my\_list = [10, 20, 30, 40]

size = len(my\_list)

The first statement assigns the list [10, 20, 30, 40] to the my\_list variable. The second statement calls the len function, passing the my\_list variable as an argument.

The function returns the value 4, which is the number of elements in the list. This value is assigned to the size variable.

The len function can be used to prevent an IndexError exception when iterating over a list with a loop. Here is an example:

my\_list = [10, 20, 30, 40]

index = 0

while index < len(my\_list):

print(my\_list[index])

index += 1

# Lists Are Mutable

Lists in Python are mutable, which means their elements can be changed. Consequently, an expression in the form list[index] can appear on the left side of an assignment operator.

The following code shows an example:

1 numbers = [1, 2, 3, 4, 5]

2 print(numbers)

3 numbers[0] = 99

4 print(numbers)

The statement in line 2 will display

[1, 2, 3, 4, 5]

The statement in line 3 assigns 99 to numbers[0]. This changes the first value in the list to 99. When the statement in line 4 executes, it will display

[99, 2, 3, 4, 5]

# Concatenating Lists

To concatenate means to join two things together. You can use the + operator to concatenate two lists. Here is an example:

list1 = [1, 2, 3, 4]

list2 = [5, 6, 7, 8]

list3 = list1 + list2

After this code executes, list1 and list2 remain unchanged, and list3 references the following list:

[1, 2, 3, 4, 5, 6, 7, 8]

You can also use the += augmented assignment operator to concatenate one list to another. Here is an example:

list1 = [1, 2, 3, 4]

list2 = [5, 6, 7, 8]

list1 += list2

The last statement appends list2 to list1. After this code executes, list2 remains

unchanged, but list1 references the following list:

[1, 2, 3, 4, 5, 6, 7, 8]

# List Slicing

A slicing expression selects a range of elements from a Sequences.

To get a slice of a list, you write an expressionin the following general format:

list\_name[start : end]

In the general format, start is the index of the first element in the slice, and end is the index marking the end of the slice. The expression returns a list containing a copy of the elements from start up to (but not including) end. For example, suppose we create the following list:

days = ['Sunday', 'Monday', 'Tuesday', 'Wednesday', 'Thursday', 'Friday', 'Saturday']

The following statement uses a slicing expression to get the elements from indexes 2 up to, but not including, 5:

mid\_days = days[2:5]

After this statement executes, the mid\_days variable references the following list:

['Tuesday', 'Wednesday', 'Thursday']

If you leave out the start index in a slicing expression, Python uses 0 as the starting index. The following interactive mode session shows an example:

1 >>> numbers = [1, 2, 3, 4, 5]

2 >>> print(numbers)

3 [1, 2, 3, 4, 5]

4 >>> print(numbers[:3])

5 [1, 2, 3]

If you leave out the end index in a slicing expression, Python uses the length of the list as the end index. The following interactive mode session shows an example:

1 >>> numbers = [1, 2, 3, 4, 5]

2 >>> print(numbers)

3 [1, 2, 3, 4, 5]

4 >>> print(numbers[2:])

5 [3, 4, 5]

If you leave out both the start and end index in a slicing expression, you get a copy of the entire list. The following interactive mode session shows an example:

1 >>> numbers = [1, 2, 3, 4, 5]

2 >>> print(numbers)

3 [1, 2, 3, 4, 5]

4 >>> print(numbers[:])

5 [1, 2, 3, 4, 5]

The slicing examples we have seen so far get slices of consecutive elements from lists. Slicing expressions can also have step value, which can cause elements to be skipped in the list.

1 >>> numbers = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]

2 >>> print(numbers)

3 [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]

4 >>> print(numbers[1:8:2])

5 [2, 4, 6, 8]

You can also use negative numbers as indexes in slicing expressions to reference positions relative to the end of the list. Python adds a negative index to the length of a list to get the position referenced by that index. The following interactive mode session shows an example:

1 >>> numbers = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]

2 >>> print(numbers)

3 [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]

4 >>> print(numbers[-5:])

5 [6, 7, 8, 9, 10]

# Finding Items in Lists with the in Operator

You can search for an item in a list using the in operator. Here is the general format of an expression written with the in operator to search for an item in a list:

item in list

In the general format, item is the item for which you are searching, and list is a list. The expression returns true if item is found in the list or false otherwise

You can use the not in operator to determine whether an item is not in a list. Here is an example:

if search not in prod\_nums:

print(search, 'was not found in the list.')

else:

print(search, 'was found in the list.')

# List Methods and Useful Built-in Functions

Lists have numerous methods that allow you to work with the elements that they contain. Python also provides some built-in functions that are useful for working with lists.

**The append Method**

The append method is commonly used to add items to a list. The item that is passed as an argument is appended to the end of the list’s existing elements.

|  |  |
| --- | --- |
| Method | Description |
| append(item) | Adds item to the end of the list. |
| index(item) | Returns the index of the first element whose value is equal to item. A ValueError exception is raised if item is not found in the list. |
| insert(index, item) | Inserts item into the list at the specified index. When an item is inserted into a list, the list is expanded in size to accommodate the new item.  If you specify an index beyond the end of the list, the item will be added to the end of the list. If you use a negative index that specifies an invalid position, the item will be inserted at the beginning of the list. |
| sort() | Sorts the items in the list so they appear in ascending order (from the lowest value to the highest value). |
| remove(item) | Removes the first occurrence of item from the list. A ValueError exception is raised if item is not found in the list. |
| reverse() | Reverses the order of the items in the list. |

**The index Method**

Earlier you saw how the in operator can be used to determine whether an item is in a list. Sometimes you need to know not only whether an item is in a list, but where it is located. The index method is useful in these cases. You pass an argument to the index method and it returns the index of the first element in the list containing that item. If the item is not found in the list, the method raises a ValueError exception.

**The insert Method**

The insert method allows you to insert an item into a list at a specific position. You pass two arguments to the insert method: an index specifying where the item should be inserted and the item that you want to insert.

**The sort Method**

The sort method rearranges the elements of a list so they appear in ascending order (from the lowest value to the highest value). Here is an example:

my\_list = [9, 1, 0, 2, 8, 6, 7, 4, 5, 3]

print('Original order:', my\_list)

my\_list.sort()

print('Sorted order:', my\_list)

When this code runs it will display the following:

Original order: [9, 1, 0, 2, 8, 6, 7, 4, 5, 3]

Sorted order: [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]

Here is another example:

my\_list = ['beta', 'alpha', 'delta', 'gamma']

print('Original order:', my\_list)

my\_list.sort()

print('Sorted order:', my\_list)

When this code runs it will display the following:

Original order: ['beta', 'alpha', 'delta', 'gamma']

Sorted order: ['alpha', 'beta', 'delta', 'gamma']

**The remove Method**

The remove method removes an item from the list. You pass an item to the method as an argument and the first element containing that item is removed. This reduces the size of the list by one element. All of the elements after the removed element are shifted one position toward the beginning of the list. A ValueError exception is raised if the item is not found in the list.

**The reverse Method**

The reverse method simply reverses the order of the items in the list. Here is an example:

my\_list = [1, 2, 3, 4, 5]

print('Original order:', my\_list)

my\_list.reverse()

print('Reversed:', my\_list)

This code will display the following:

Original order: [1, 2, 3, 4, 5]

Reversed: [5, 4, 3, 2, 1]

# The del Statement

The remove method that you saw earlier removes a specific item from a list, if that item is in the list. Some situations might require that you remove an element from a specific index, regardless of the item that is stored at that index. This can be accomplished with the del statement. Here is an example of how to use the del statement:

my\_list = [1, 2, 3, 4, 5]

print('Before deletion:', my\_list)

del my\_list[2]

print('After deletion:', my\_list)

This code will display the following:

Before deletion: [1, 2, 3, 4, 5]

After deletion: [1, 2, 4, 5]

# The min and max Functions

Python has two built-in functions named min and max that work with sequences. The min function accepts a sequence, such as a list, as an argument and returns the item that has the lowest value in the sequence. Here is an example:

my\_list = [5, 4, 3, 2, 50, 40, 30]

print('The lowest value is', min(my\_list))

This code will display the following:

The lowest value is 2

The max function accepts a sequence, such as a list, as an argument and returns the item that has the highest value in the sequence. Here is an example:

my\_list = [5, 4, 3, 2, 50, 40, 30]

print('The highest value is', max(my\_list))

This code will display the following:

The highest value is 50

# 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>
* IBM. (2003, August 27). Expresiones aritméticas. <https://publib.boulder.ibm.com/wcmid/mp/v50/helpsystem/es/ct000102.html>

**CONCLUSIÓN:**

En conclusión considero que estas herramientas son muy interesantes, y útiles a la hora de hacer código en Python, mientras leía trataba de buscarles usos en situaciones de la vida diaria, en la automatización de procesos o hasta en los videojuegos, por ello no puedo esperar a poner las en practica y ver que puedo hacer con estos nuevos conocimientos.