8 List

In Python a list is a collection of items in a particular order. You can make a list that includes the letters of the alphabet, the digits from 0-9, or the names of all the people in your family. You can put anything you want into a list, and the items in your list don't have to be related in any particular way. Because a list usually contains more than one element, it's a good idea to make the name of your list plural, such as letters, digits, or names. In Python, square brackets [] indicate a list, and individual elements in the list are separated by commas.

Example

```
bicycles = ["trek", "cannondale", "redline", "specialized"] # list of bicycles,
    each element is a string
print(bicycles)
```

The above example creates a list of bicycles and it elements are all strings. However, you can store any type of data in a list. For example, you can store numbers in a list, and you can even mix different types of data in a list. For example, you can store a string, a number, and a Boolean value in the same list. The following example shows a list that includes several different types of data:

```
mix = ["hello", 1, True]
print(mix) # ['hello', 1, True]
```

8.1 Creating a List

You can create a list by using square brackets [] and separating the elements in the list with commas. The following example shows how to create a list:

```
bicycles = ["trek", "cannondale", "redline", "specialized"]
print(bicycles) # ['trek', 'cannondale', 'redline', 'specialized']
```

You can also create an empty list and add elements to it later. The following example shows how to create an empty list and add elements to it later:

```
bicycles = [] # create an empty list
bicycles.append("trek") # add an element to the list
bicycles.append("cannondale")
bicycles.append("redline")
bicycles.append("specialized")
print(bicycles) # ['trek', 'cannondale', 'redline', 'specialized']
```

8.2 Accessing Elements in a List

Similar to string, you can access elements in a list by using the index of the element. The index of the first element in a list is 0, the index of the second element is 1, and so on. The following example shows how to access elements in a list:

```
bicycles = ["trek", "cannondale", "redline", "specialized"]
print(bicycles[0]) # trek
```

You can also use negative indices to access elements in a list. The index -1 refers to the last item in a list, the index -2 refers to the second last item in a list, and so on. The following example shows how to access elements in a list using negative indices:

```
bicycles = ["trek", "cannondale", "redline", "specialized"]
print(bicycles[-1]) # specialized
```

Slice a List

You can also use a range of indices to access a subset of elements in a list. The following example shows how to access a subset of elements in a list:

```
bicycles = ["trek", "cannondale", "redline", "specialized"]
print(bicycles[0:2]) # ['trek', 'cannondale']
```

8.3 Changing, Adding, and Removing Elements

You can modify elements in a list, add new elements to a list, and remove elements from a list. You can modify elements in a list by using the index of the element you want to change.

Modifying Elements in a List

```
bicycles = ["trek", "cannondale", "redline", "specialized"]
bicycles[0] = "ducati" # change the first element in the list
print(bicycles) # ['ducati', 'cannondale', 'redline', 'specialized']
```

Adding Elements to a List

You can add elements to the end of a list by using the append() method. The following example shows how to add elements to the end of a list:

```
bicycles = ["trek", "cannondale", "redline", "specialized"]
bicycles.append("ducati") # add ducati to the end of the list
print(bicycles) # ['trek', 'cannondale', 'redline', 'specialized', 'ducati']
```

Inserting Elements into a List

You can add a new element at any position in your list by using the insert() method. You do this by specifying the index of the new element and the value of the new item. The following example shows how to insert an element into a list:

```
bicycles = ["trek", "cannondale", "redline", "specialized"]
bicycles.insert(0, "ducati") # insert ducati at the beginning of the list
print(bicycles) # ['ducati', 'trek', 'cannondale', 'redline', 'specialized']
```

Difference of append() and insert()

The difference between append() and insert() is that append() adds a new element to the end of a list, whereas insert() adds a new element at any position in your list. The following example shows the difference between append() and insert():

```
bicycles = ["trek", "cannondale", "redline", "specialized"]
bicycles.append("ducati") # add ducati to the end of the list
bicycles.insert(0, "ducati") # insert ducati at the beginning of the list
print(bicycles) # ['ducati', 'trek', 'cannondale', 'redline', 'specialized', '
    ducati']
```

Removing Elements from a List

You can remove an item according to its position in the list by using the del statement, if you know the position of the item you want to remove. The following example shows how to remove an element from a list:

```
bicycles = ["trek", "cannondale", "redline", "specialized"]
del bicycles[0] # remove the first element in the list
print(bicycles) # ['cannondale', 'redline', 'specialized']
```

Removing an Item Using the pop() Method

You can also remove an item according to its position in the list by using the pop() method. When you use the pop() method, the item you want to remove is no longer stored in the list. The pop() method removes the last item in a list, but it lets you work with that item after removing it. The following example shows how to remove an element from a list using the pop() method:

```
bicycles = ["trek", "cannondale", "redline", "specialized"]
popped_bicycle = bicycles.pop() # remove the last element in the list
print(bicycles) # ['trek', 'cannondale', 'redline']
print(popped_bicycle) # specialized
```

Popping Items from any Position in a List

You can use the pop() method to remove an item in a list at any position by including the index of the item you want to remove in parentheses. The following example shows how to remove an element from a list using the pop() method:

```
bicycles = ["trek", "cannondale", "redline", "specialized"]
first_owned = bicycles.pop(0) # remove the first element in the list
print(bicycles) # ['cannondale', 'redline', 'specialized']
print(first_owned) # trek
```

Removing an Item by Value

You can remove an item according to its value by using the remove() method. The following example shows how to remove an element from a list using the remove() method:

```
bicycles = ["trek", "cannondale", "redline", "specialized"]
bicycles.remove("trek") # remove the element with value "trek"
print(bicycles) # ['cannondale', 'redline', 'specialized']
```

Note that the remove() method deletes only the first occurrence of the value you specify. If there's a possibility the value appears more than once in the list, you'll need to use a loop to make sure all occurrences of the value are removed.

```
bicycles = ["trek", "cannondale", "redline", "specialized", "trek"]
while "trek" in bicycles: # remove all occurrences of "trek"
    bicycles.remove("trek")
print(bicycles) # ['cannondale', 'redline', 'specialized']
```

8.4 Organizing a List

You can organize a list in alphabetical order, reverse alphabetical order, or in the order you added items to the list. You can also reverse the order of a list permanently.

Sorting a List Permanently with the sort() Method

The sort() method changes the order of a list permanently. If the list is a number list, the sort() method will sort the list in ascending order. If the list is a string list, the sort() method will sort the list in alphabetical order. The following example shows how to sort a number list and a string list using the sort() method:

```
numbers = [3, 1, 4, 1, 5, 9, 2, 6, 5, 3]
numbers.sort() # sort the list in ascending order
print(numbers) # [1, 1, 2, 3, 3, 4, 5, 5, 6, 9]
cars = ["bmw", "audi", "toyota", "subaru"]
cars.sort() # sort the list in alphabetical order
print(cars) # ['audi', 'bmw', 'subaru', 'toyota']
```

Sorting a List in Reverse Alphabetical Order

You can also sort a list in reverse alphabetical order by passing the argument reverse=True to the sort() method. The following example shows how to sort a list in reverse alphabetical order using the sort() method:

```
cars = ["bmw", "audi", "toyota", "subaru"]
cars.sort(reverse=True) # sort the list in reverse alphabetical order
print(cars) # ['toyota', 'subaru', 'bmw', 'audi']
```

Sorting a List Temporarily with the sorted() Function

The sorted() function lets you display your list in a particular order but doesn't affect the actual order of the list. The following example shows how to sort a list temporarily using the sorted() function:

```
cars = ["bmw", "audi", "toyota", "subaru"]
print("Here is the original list:")
print(cars) # ['bmw', 'audi', 'toyota', 'subaru']
print("\nHere is the sorted list:")
print(sorted(cars)) # ['audi', 'bmw', 'subaru', 'toyota']
print("\nHere is the original list again:")
print(cars) # ['bmw', 'audi', 'toyota', 'subaru']
```

Printing a List in Reverse Order

You can reverse the original order of a list permanently by using the reverse() method. The following example shows how to reverse a list permanently using the reverse() method:

```
cars = ["bmw", "audi", "toyota", "subaru"]
cars.reverse() # reverse the list permanently
print(cars) # ['subaru', 'toyota', 'audi', 'bmw']
```

Finding the Length of a List

Same as strings, you can find the length of a list by using the len() function. The following example shows how to find the length of a list using the len() function:

```
cars = ["bmw", "audi", "toyota", "subaru"]
print(len(cars)) # 4
```

8.5 Avoiding Index Errors When Working with Lists

If you try to access an item in a list that doesn't exist, you'll get an index error. For example, if you try to access the third item in a list that has only two items, you'll get an index error. The following example shows how to avoid index errors when working with lists:

```
motorcycles = ["honda", "yamaha", "suzuki"]
print(motorcycles[3]) # IndexError: list index out of range
```

8.6 Looping Through an Entire List

You can loop through the entire list by using a for loop. The following example shows how to loop through an entire list using a for loop:

```
magicians = ["alice", "david", "carolina"]
for magician in magicians: # The in keyword tells Python to pull a value from
    the list magicians and store it in the variable magician.
    print(magician)
```