Lists

Lists are one of the most essential data structures in Python. They are used to store multiple items in a single variable. A list is created by putting elements inside square brackets (`[]`), separated by commas (`,`). Lists are mutable, meaning that you can change the elements of a list after it has been created.

Here's an example of creating a list in Python:

```
fruits = ["apple", "banana", "cherry", "orange"]
print(fruits)
```

In the example above, fruits is a list of strings, but lists can contain elements of any data type, including integers, floats, and even other lists.

Accessing Elements of a List

You can access individual elements in a list by using the index of the element, which is an integer that specifies the position of the element in the list. Indexing in Python starts from 0, so the first element in the list has an index of 0, the second element has an index of 1, and so on.

Here's an example of accessing elements in a list:

```
fruits = ["apple", "banana", "cherry", "orange"]
print(fruits[0])
print(fruits[1])
print(fruits[2])
print(fruits[3])
```

You can also use negative indexing to access elements from the end of the list. The last element in the list has an index of -1, the second-to-last element has an index of -2, and so on.

```
fruits = ["apple", "banana", "cherry", "orange"]
print(fruits[-1])
print(fruits[-2])
print(fruits[-3])
print(fruits[-4])
```

Modifying Elements of a List

You can change the elements in a list by assigning new values to them. You can do this by using the index of the element you want to change and the assignment operator (`=`).

```
fruits = ["apple", "banana", "cherry", "orange"]
fruits[1] = "kiwi"
print(fruits)
```

Adding Elements to a List

You can add elements to a list using the `append()` method, which adds an element to the end of the list.

```
fruits = ["apple", "banana", "cherry", "orange"]
fruits.append("pear")
print(fruits)
```

You can also add elements to a list using the `insert()` method, which inserts an element at a specified position in the list.

```
fruits = ["apple", "banana", "cherry", "orange"]
fruits.insert(1, "kiwi")
print(fruits)
```

Removing Elements from a List

You can remove elements from a list using the `remove()` method, which removes the first occurrence of the specified element from the list.

```
fruits = ["apple", "banana", "cherry", "orange"]
fruits.remove("banana")
print(fruits)
```

You can also remove elements from a list using the `pop()` method, which removes the element at a specified position in the list and returns the removed element. If no index is specified, `pop()` removes the last element in the list.

```
fruits = ["apple", "banana", "cherry", "orange"]
removed_fruit = fruits.pop(1)
print(fruits)
print("Removed fruit:", removed_fruit)
```

Sorting a List

You can sort the elements in a list using the `sort()` method, which sorts the elements in ascending order by default.

```
fruits = ["cherry", "orange", "apple", "banana"]
fruits.sort()
print(fruits)
```

You can sort the elements in descending order by specifying the `reverse=True` argument in the `sort()` method.

```
fruits = ["cherry", "orange", "apple", "banana"]
fruits.sort(reverse=True)
print(fruits)
```

List Length

You can find the number of elements in a list using the `len()` function.

```
fruits = ["apple", "banana", "cherry", "orange"]
num_fruits = len(fruits)
print("Number of fruits:", num_fruits)
```

List of Lists

A list of lists is a data structure that contains multiple lists within a single list. This data structure is useful when you want to store a collection of items that can be grouped together, such as a list of students and their grades, a list of items in a grocery store and their prices, etc.

Here's an example of how you can create a list of lists in Python:

```
students = [['John', 80, 85, 90], ['Jane', 75, 80, 85], ['Jim', 70, 75, 80]]
print(students)
```

In the example above, students is a list of lists where each inner list represents a student and their grades.

You can access elements in a list of lists by using nested indices. The first index specifies the inner list, and the second index specifies the element within that inner list.

```
students = [['John', 80, 85, 90], ['Jane', 75, 80, 85], ['Jim', 70, 75, 80]]
print(students[0][0])
print(students[1][1])
print(students[2][2])
```

You can use loops to iterate over each list in a list of lists.

```
students = [['John', 80, 85, 90], ['Jane', 75, 80, 85], ['Jim', 70, 75, 80]]

for student in students:
    print("Student Name:", student[0])
    print("Grades:", student[1:])
```

In the example above, the loop iterates over the inner lists of the students list, and the student variable holds the current inner list. The inner lists can further be iterated over using a nested loop. For example:

```
students = [['John', 80, 85, 90], ['Jane', 75, 80, 85], ['Jim', 70, 75, 80]]

for student in students:
    print("Student Name:", student[0])
    for grade in student[1:]:
        print(grade)
```

In the example above, the first loop iterates over the inner lists of the students list, and the student variable holds the current inner list. The second loop iterates over the elements relating to grade within each inner list.

You can use the `len()` function to find the number of inner lists in a list of lists.

```
students = [['John', 80, 85, 90], ['', 75, 80, 85], ['Jim', 70, 75, 80]]

num_students = len(students)
```

```
print("Number of students:", num_students)
```

List Comprehensions

List comprehensions provide a concise and powerful way to create new lists from existing lists. It allows you to create a new list by applying an expression to each element of an existing list.

Here's the syntax for a list comprehension:

```
new_list = [expression for item in iterable if condition]
```

- `expression` is the expression that will be applied to each element in the iterable.
- `item` is a variable that represents each element in the iterable.
- `iterable` is the iterable that contains the elements to be transformed.
- `condition` is an optional condition that filters the elements of the iterable based on some criteria.

For example, let's say you want to create a new list that contains the squares of the numbers from 1 to 10. You can do this using a list comprehension as follows:

```
squares = [x**2 for x in range(1, 11)]
print(squares)
```

The example below create a new list that contains only the even numbers from a given list:

```
numbers = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
evens = [x for x in numbers if x % 2 == 0]
print(evens)
```

Exercises

1. Create a list of your favorite foods and assign it to a variable called `favorite_foods`.
2. Print the length of your `favorite_foods` list.
3. Print the second to last element in your `favorite_foods` list.
4. Sort your `favorite_foods` list in reverse alphabetical order.
5. Modify the first element in your `favorite_foods` list to be a different food.
6. Add a new food to the end of your `favorite_foods` list.

7. Remove the second element in your `favorite_foods` list.
8. Create a list of lists called `people` , where each inner list represents a person's name, age and favorite food.
9. Append a new person to the `people` list.
10. Print the names of the people in the `people` list.
11. Calculate the average age of the people in the `people` list.
12. Concatenate the favorite foods of the people in the `people` list into a single string called `people_favorite_foods`.
13. Create a new list that contains the squares of the even numbers from a given list.
14. Create a new list that contains only the uppercase letters from a given string.
15. Create a new list that contains the length of each word in a given sentence.

Project: Grocery List

Project Description

Create a Python program that acts as a grocery list. The program should allow the user to add and remove items from the list, and display the current list at any time. The program should also allow the user to check whether a particular item is on the list.

Project Requirements

1. The program should allow the user to add and remove items from the list.
2. The program should display the current list at any time.
3. The program should allow the user to check whether a particular item is on the list.
4. The program should handle errors appropriately, for example, when the user tries to remove an item that is not on the list.

Example Output

```
Welcome to Grocery List!
1. Add an item
2. Remove an item
3. Check if an item is on the list
4. Display the current list
5. Quit
Enter your choice: 1

Enter the item to add: Milk
Milk has been added to the list.

Enter your choice: 1

Enter the item to add: Eggs
Eggs has been added to the list.

Enter your choice: 4

Current list: ['Milk', 'Eggs']

Enter your choice: 3

Enter the item to check: Milk
Milk is on the list.

Enter your choice: 2

Enter the item to remove: Bread
Error: Bread is not on the list.

Enter your choice: 2

Enter the item to remove: Milk
Milk has been removed from the list.

Enter your choice: 4

Current list: ['Eggs']

Enter your choice: 5
```

Further Reading

Check out these following resources to learn more about lists in Python:

- W3Schools
  - Python Lists
  - Python List Methods
- Real Python
  - Python Lists and Tuples
- GeeksforGeeks
  - Python Lists
- Python Documentation
  - Python List Methods

Additionally, you can use the search terms "Python lists" or "Python list methods" to find more resources on the internet.