7. Lists

Creating a List

To create a list in Python, you enclose a comma-separated sequence of items inside square brackets `[ ]`. Here's an example:

```
fruits = ['apple', 'banana', 'cherry']
numbers = [1, 2, 3, 4, 5]
mixed = ['hello', 42, True, 3.14]
```

You can also create an empty list and add elements to it later:

```
empty_list = []
```

Accessing List Elements

Each element in a list is associated with an index, which is a numerical value representing its position within the list. Indexing in Python starts from 0, meaning the first element has an index of 0, the second element has an index of 1, and so on.

You can access individual elements using their index inside square brackets:

```
fruits = ['apple', 'banana', 'cherry']
print(fruits[0])  # Output: 'apple'
print(fruits[2])  # Output: 'cherry'
```

You can also use negative indices to access elements from the end of the list:

```
fruits = ['apple', 'banana', 'cherry']
print(fruits[-1])  # Output: 'cherry'
print(fruits[-3])  # Output: 'apple'
```

Modifying List Elements

One of the key features of lists is that they are mutable, meaning you can modify or change their elements after creation. You can assign a new value to an existing index like this:

```python
fruits = ['apple', 'banana', 'cherry']
fruits[1] = 'orange'
print(fruits)  # Output: ['apple', 'orange', 'cherry']
```

List Slicing

Python lists support slicing, which allows you to create a new list by extracting a subset of elements from an existing list. The syntax for slicing is `list[start:stop:step]`, where `start` is the index to start from (inclusive), `stop` is the index to stop at (exclusive), and `step` is the step size (optional, defaults to 1).

```python
fruits = ['apple', 'banana', 'cherry', 'date', 'elderberry']
print(fruits[1:4])  # Output: ['banana', 'cherry', 'date']
print(fruits[:3])   # Output: ['apple', 'banana', 'cherry']
print(fruits[2:])   # Output: ['cherry', 'date', 'elderberry']
print(fruits[::2])  # Output: ['apple', 'cherry', 'elderberry']
```

List Methods

Python provides various built-in methods that you can use to manipulate lists. Here are some commonly used methods:

- `append(element)`: Adds an element to the end of the list.
- `insert(index, element)`: Inserts an element at the specified index.
- `remove(element)`: Removes the first occurrence of the specified element from the list.
- `pop([index])`: Removes and returns the element at the specified index (or the last element if no index is given).
- `index(element[, start[, end]])`: Returns the index of the first occurrence of the specified element within the optional range `start` to `end`.
- `count(element)`: Returns the number of times the specified element appears in the list.
- `sort([reverse=True|False])`: Sorts the list in ascending (default) or descending order.
- `reverse()`: Reverses the order of the list elements.
- `extend(iterable)`: Adds all elements from an iterable (e.g., list, tuple, string) to the end of the list.
- `clear()`: Removes all elements from the list.

Here are some examples:

```python
fruits = ['apple', 'banana', 'cherry']
fruits.append('date')
print(fruits)  # Output: ['apple', 'banana', 'cherry', 'date']

fruits.insert(2, 'orange')
print(fruits)  # Output: ['apple', 'banana', 'orange', 'cherry', 'date']

fruits.remove('banana')
print(fruits)  # Output: ['apple', 'orange', 'cherry', 'date']

last_fruit = fruits.pop()
print(last_fruit)  # Output: 'date'
print(fruits)  # Output: ['apple', 'orange', 'cherry']

index = fruits.index('orange')
print(index)  # Output: 1

fruits.reverse()
print(fruits)  # Output: ['cherry', 'orange', 'apple']

numbers = [5, 2, 8, 1, 9]
numbers.sort()
print(numbers)  # Output: [1, 2, 5, 8, 9]
```

List Comprehensions

Python offers a concise way to create lists using list comprehensions. A list comprehension is an expression that creates a new list by iterating over an existing iterable (e.g., list, tuple, string) and applying an operation to each item.

The basic syntax for a list comprehension is:

```python
new_list = [expression for item in iterable]
```

Here's an example:

```python
numbers = [1, 2, 3, 4, 5]
squared_numbers = [x ** 2 for x in numbers]
print(squared_numbers)  # Output: [1, 4, 9, 16, 25]
```

You can also include conditional statements in list comprehensions:

```
numbers = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
even_numbers = [num for num in numbers if num % 2 == 0]
print(even_numbers)  # Output: [2, 4, 6, 8, 10]
```

List comprehensions can be more concise and easier to read than using loops or other constructs for creating lists, especially for simple cases.

See Also
8. List Methods