3. Methods

String Methods

Strings are a sequence of characters, and Python provides several built-in methods to manipulate and work with them. Here are some commonly used string methods:

1. `str.upper()` and `str.lower()`: These methods return a new string where all characters are converted to uppercase or lowercase, respectively.

```
name = "Alice"
print(name.upper())  # Output: ALICE
print(name.lower())  # Output: alice
```

2. `str.strip()`: This method returns a new string with leading and trailing whitespace characters removed.

```
text = "  Hello, World!  "
print(text.strip())  # Output: "Hello, World!"
```

3. `str.split(separator)`: This method splits the string into a list of substrings using the specified separator.

```
sentence = "This is a sample sentence."
words = sentence.split()  # Split on whitespace
print(words)  # Output: ['This', 'is', 'a', 'sample', 'sentence.']
```

4. `str.replace(old, new)`: This method returns a new string where all occurrences of the `old` substring are replaced with the `new` substring.

```
original = "Hello, World!"
modified = original.replace("World", "Python")
print(modified)  # Output: "Hello, Python!"
```

List Methods

Lists are ordered collections of items, and Python provides several built-in methods to work with them.

1. `list.append(item)`: This method adds the `item` to the end of the list.

```
fruits = ["apple", "banana"]
fruits.append("orange")
print(fruits)  # Output: ['apple', 'banana', 'orange']
```

2. `list.remove(item)`: This method removes the first occurrence of `item` from the list.

```
numbers = [1, 2, 3, 2, 4]
numbers.remove(2)
print(numbers)  # Output: [1, 3, 2, 4]
```

3. `list.sort()`: This method sorts the items in the list in ascending order.

```
scores = [85, 92, 77, 68, 94]
scores.sort()
print(scores)  # Output: [68, 77, 85, 92, 94]
```

4. `list.reverse()`: This method reverses the order of the items in the list.

```
numbers = [1, 2, 3, 4, 5]
numbers.reverse()
print(numbers)  # Output: [5, 4, 3, 2, 1]
```

Dictionary Methods

Dictionaries are unordered collections of key-value pairs, and Python provides several built-in methods to work with them.

1. `dict.keys()`: This method returns a view object containing the keys of the dictionary.

```
person = {"name": "Alice", "age": 25, "city": "New York"}
keys = person.keys()
print(keys)  # Output: dict_keys(['name', 'age', 'city'])
```

2. `dict.values()`: This method returns a view object containing the values of the dictionary.

```
person = {"name": "Alice", "age": 25, "city": "New York"}
values = person.values()
print(values)  # Output: dict_values(['Alice', 25, 'New York'])
```

3. **`dict.get(key, default=None)`**: This method returns the value associated with the specified `key`. If the `key` is not found, it returns the `default` value (or `None` if not specified).

```
person = {"name": "Alice", "age": 25, "city": "New York"}
name = person.get("name")
phone = person.get("phone", "Not available")
print(name)  # Output: "Alice"
print(phone)  # Output: "Not available"
```

4. **`dict.update(other_dict)`**: This method updates the dictionary with the key-value pairs from `other_dict`.

```
person = {"name": "Alice", "age": 25}
new_info = {"city": "New York", "job": "Engineer"}
person.update(new_info)
print(person)  # Output: {'name': 'Alice', 'age': 25, 'city': 'New
York', 'job': 'Engineer'}
```

See Also 4. String Operations]