Strings, Lists, and Dictionaries - MAD 102 Week 7 Notes

Hia Al Saleh

October 16th, 2024

Contents

1 Strings

Strings are sequences of characters, used for storing text data. They are immutable, meaning once a string is created, it cannot be modified in place.

1.1 Basic String Operations

```
# Concatenation
greeting = "Hello" + " " + "World"
print(greeting)  # Output: Hello World

# Repetition
print("Hi! " * 3)  # Output: Hi! Hi! Hi!
```

1.2 Slicing and Stride

```
text = "PythonProgramming"
print(text[0:6])    # Output: Python
print(text[::2])    # Output: PtoPormig (Every 2nd
    character)
print(text[::-1])   # Output: gnimmargorPnohtyP (Reversed)
```

1.3 Common String Methods

- `strip()` removes leading and trailing whitespace.
- `replace()` replaces occurrences of a substring.
- `find()` returns the index of a substring or -1 if not found.

```
name = "  Alice  "
print(name.strip())  # Output: Alice

sentence = "I love Python!"
print(sentence.replace("Python", "coding"))  # I love
    coding!
```

Useful String Methods:

- `replace()`: Replace a substring with another.
- `find()`, `rfind()`: Locate the position of a substring.
- `split()`: Split a string into a list of tokens based on a separator.
- `join()`: Join a list of strings with a specified separator.
- `capitalize()`, `lower()`, `upper()`, `title()`, `strip()`: Modify string casing or remove extra spaces.

2 Lists

Lists are ordered collections of items that are mutable, meaning their contents can be modified.

2.1 Creating Lists

```
# Using square brackets
numbers = [1, 2, 3, 4, 5]

# Using the list() function
chars = list("hello")  # ['h', 'e', 'l', 'l', 'o']
```

2.2 Adding and Removing Elements

- `append()` adds an element to the end of the list.
- `insert()` inserts an element at a specific index.
- `remove()` removes the first occurrence of an element.

```
fruits = ["apple", "banana", "cherry"]
fruits.append("orange")
fruits.insert(1, "grape")
print(fruits)  # ['apple', 'grape', 'banana', 'cherry', '
    orange']
fruits.remove("banana")
print(fruits)  # ['apple', 'grape', 'cherry', 'orange']
```

2.3 Sorting and Reversing Lists

```
numbers = [4, 2, 9, 1]
numbers.sort()  # In-place sorting
print(numbers)  # [1, 2, 4, 9]

numbers.reverse()
print(numbers)  # [9, 4, 2, 1]
```

2.4 Nested Lists and List Comprehensions

```
# Nested List
matrix = [[1, 2], [3, 4], [5, 6]]
print(matrix[1][0])  # Output: 3

# List Comprehension
```

```python
squares = [x**2 for x in range(5)]
print(squares)  # [0, 1, 4, 9, 16]
```

3 Dictionaries

Dictionaries are key-value pairs that allow efficient lookups. They are mutable and unordered in versions prior to Python 3.7.

3.1 Creating and Accessing Dictionaries

```python
# Using curly braces
person = {"name": "Alice", "age": 25}

# Accessing values
print(person["name"])  # Output: Alice

# Adding a new key-value pair
person["location"] = "New York"
print(person)  # {'name': 'Alice', 'age': 25, 'location':
    'New York'}
```

Common Dictionary Operations:

- Add/modify entries: `dict[key] = value`.
- Remove entries: `del dict[key]` or `pop()`.
- Check for a key: `key in dict`.

3.2 Modifying and Removing Entries

```python
# Modify a value
person["age"] = 30

# Remove a key-value pair
del person["location"]
print(person)  # {'name': 'Alice', 'age': 30}
```

3.3 Dictionary Methods and Iteration

- `get()` safely retrieves a value.
- `keys()`, `values()`, and `items()` allow iteration.

```python
# Safely access a key
print(person.get("age", "Not Found"))  # Output: 30

# Iterate over keys and values
for key, value in person.items():
    print(f"{key}: {value}")

# Output:
# name: Alice
# age: 30
```

3.4 Nested Dictionaries and Merging

```python
# Nested Dictionary
company = {
    "employee1": {"name": "Alice", "age": 30},
    "employee2": {"name": "Bob", "age": 25}
}
print(company["employee1"]["name"])  # Output: Alice

# Merging Dictionaries
dict1 = {"a": 1, "b": 2}
dict2 = {"b": 3, "c": 4}
merged = {**dict1, **dict2}
print(merged)  # {'a': 1, 'b': 3, 'c': 4}
```

4 Class Exercises

4.1 Longest Word Finder Program

This program keeps asking for words until the user decides they are done, and then identifies the longest word entered. It also checks if two words have the same length and prints a message in that case.

4.1.1 Code

```
def longest_word():
    longest = ""
    while True:
        word = input("Enter a word (or type 'done' to
            finish): ").strip()
        if word.lower() == "done":
            break

        if len(word) > len(longest):
            longest = word
        elif len(word) == len(longest):
            print(f"'{word}' and '{longest}' have the same
                 length.")

    print(f"The longest word entered is: '{longest}'")

# Run the longest word finder
longest_word()
```

4.1.2 Explanation

1. The program takes input in a loop until the user types `"done"`.
2. It compares the length of each word with the current longest word and updates it if necessary.
3. If two words have the same length, it prints a message indicating that.
4. The `.strip()` method ensures that any leading or trailing spaces are removed from the input.

4.2 Password Validator Program

This program checks if a given password meets certain rules: it must be at least 10 characters long and contain only letters and numbers.

4.2.1 Code

```python
def is_valid_password(password):
    if len(password) < 10:
        return False

    if not password.isalnum():
        return False

    return True

def password_check():
    password = input("Enter your password: ").strip()
    if is_valid_password(password):
        print("Password is valid.")
    else:
        print("Invalid password. Ensure it is at least 10
            characters and contains only letters and
            numbers.")

# Run the password check
password_check()
```

4.2.2 Explanation

1. The `is_valid_password()` function checks if the password meets both of the following conditions:
   - It has at least 10 characters.
   - It contains only alphanumeric characters (letters and numbers), with no spaces or special symbols.
2. The `password_check()` function takes input from the user and calls `is_valid_password()` to validate the password.