Python Week 2 Review: Comprehensive Study Guide

This summary covers the key Python programming concepts discussed in the video, including data types, list and string slicing, control flow, loops, built-in methods, and mutability.

1. Core Data Types and Classes

Python uses specific data types to define the nature of an object. The type() function can be used to check an object's class.

- **Integers (int):** Whole numbers that can be zero, positive, or negative.
- **Floats (float):** Numbers containing decimal places, which can also be zero, positive, or negative.
- **Strings (str):** A sequence of characters enclosed in single or double quotes. Each character (including spaces) acts as an individual element.
- **Lists (list):** An ordered sequence of objects enclosed in **straight brackets []**. Lists can hold any data type—integers, floats, strings, and even other lists nested inside them.

---

2. Zero-Indexing and Indexing Basics

- All sequences in Python (lists, strings, etc.) are **zero-indexed**, meaning the very first element is at index 0.
- **Positive Indexing:** Counts forward from the beginning, starting at 0.
- **Negative Indexing:** Counts backward from the end of the sequence.
    - Index -1 refers to the last element.
    - Index -2 refers to the second-to-last element, and so on.

---

3. Slicing Syntax (Lists and Strings)

Slicing allows you to extract a specific range of elements from a list or string using the straight bracket syntax:
Python
```
sequence[start:stop:step]
```

Key Slicing Rules:

- **Start:** The index where the slice begins (inclusive).
- **Stop:** The index where the slice ends (**exclusive**; Python does not include this element).
- **Step:** The increment value determining how Python steps through the sequence. By default, the step is 1.

- **Massive/Out-of-Bounds Stops:** If the stop index exceeds the length of the list, Python automatically caps it at the end of the sequence without throwing an error.
- **Omitting Values:**
    - Leaving the start blank defaults to the beginning (0).
    - Leaving the stop blank defaults to the end of the sequence.
    - [:] copies the entire sequence.
- **Reversing:** A step of -1 with empty start/stop fields (e.g., sequence[::-1]) reverses the sequence.

---

4. Loops and Control Flow

Loops are used to iterate over a sequence or run code while a condition is met.

For Loops

- **Iterating by Element:** Directly loops through each item in a sequence.

  Python
  ```
  for element in my_list:
      print(element)
  ```
- **Iterating by Index (with range()):** Loops through a generated set of numbers up to, but not including, the stop value. Combined with len(), it loops through every index of a sequence.

  Python
  ```
  for i in range(len(my_list)):
      print(i, my_list[i])
  ```

While Loops

- Evaluates a conditional statement and continuously executes the indented code block as long as the condition remains True.

Modulo Operator (%)

- Returns the remainder of a division operation. Commonly used to check if a number is even (x % 2 == 0) or odd (x % 2 != 0).

---

5. Built-in List Methods

List methods dynamically modify or evaluate lists. Note that these methods alter the list *in place* because lists are mutable.

- **.append(element)**: Adds a single item directly to the very end of the list.
- **.extend(iterable)**: Appends elements from another iterable sequence (like a list or tuple) one-by-one to the end of the target list.

- **.insert(index, element)**: Inserts a specified item at a given index, pushing all subsequent items up.
- **.pop(index)**: Removes and returns the element at the specified index. If no index is provided, it automatically removes the last element.
- **.remove(value)**: Deletes the **first instance** of the specified value from the list.
- **.index(value)**: Returns the index position of the first instance of a specified value.
- **.count(value)**: Counts how many times a given value appears in the list.
- **.sort()**: By default, sorts the elements of the list in increasing/ascending order. Note: You cannot sort a list containing incompatible data types (e.g., mixing strings and numbers).
- **sum(list)**: A built-in function that adds all numerical values inside a list together.

---

6. Built-in String Methods

String methods return **new values** rather than updating the original string because strings are immutable.

- **.upper()**: Converts all characters in the string to uppercase.
- **.lower()**: Converts all characters in the string to lowercase.
- **.title()**: Capitalizes the first letter of every word while keeping the rest lowercase.
- **.capitalize()**: Capitalizes only the very first character of the entire string.
- **.strip()**: Removes leading and trailing whitespaces from both ends of the string. Use .lstrip() for the left side only and .rstrip() for the right side only.
- **.replace(old, new)**: Replaces **every instance** of a specified substring with a new substring.
- **.find(substring)**: Searches for a substring and returns the starting index of its **first occurrence**. If the substring is not found, it returns -1.
- **.count(substring)**: Counts the total number of times a substring appears throughout the string.

---

7. Concept Check: Mutability vs. Immutability

Understanding the distinction between mutable and immutable structures is vital for programming in Python:

Property	Mutable Objects (e.g., Lists)	Immutable Objects (e.g., Strings)
Definition	The object can be modified after it is created.	The object cannot be altered once created.
Direct Assignment	Allowed. Modifying an item via list[0] = 'x' works perfectly.	Prohibited. Attempting string[0] = 'x' results in a TypeError.
Behavior of Methods	Methods alter the original object directly in place.	Methods create and return a brand new object; the original stays unchanged.

Sources:

- 2026-07-26 20-04-50.mp4