📚 Python Strings Basics

Brief Overview

This note covers **Python strings** and was created from the Lecture 3 : Beginner Friendly Python Strings | Everything You Need to Know |#pythonprogramming #2025 YouTube video. It delves into data types, string definition, concatenation, slicing, and immutability, all tailored for beginners.

Key Points

- Grasp integer, float, and string fundamentals.
- Learn string concatenation and escape character usage.
- Master indexing, slicing, and reverse slicing techniques.
- Understand string immutability and the len() function.

📚 Data Types in Python

🧮 Integer

> **Integer** – a whole number without a fractional part.

- Created by writing the number directly, e.g., 3.
- Recognized by type() as .

🌡️ Float

> **Float** – a number that contains a decimal point (fractional part).

- Example: 3.14.
- Recognized by type() as .

🧵 String

> **String** – a sequence of characters enclosed in single ('...') or double ("...") quotes.

- Example: "hello" or 'hello'.
- Without quotes, hello raises a **NameError** because it isn't a defined identifier.
- Recognized by type() as .

Data Type	Syntax Example	type() Result
Integer	3	
Float	3.14	
String	"hello"	

🔧 Working with Strings

✨ Defining Strings

- Enclose text in matching quotes: "text" or 'text'.
- Both single and double quotes work, but the same type must open and close the string.

➕ Concatenation

> **Concatenation** – joining two strings end-to-end using the + operator.

```
result = "hello" + "2"   # result → "hello2"
```

- Adding a non-string (e.g., an integer) to a string causes a **TypeError**.
- Convert non-strings to strings first, or enclose them in quotes.

🚫 Errors with Mismatched Types

- "hello" + 2 → **TypeError** (int cannot be concatenated with str).
- Use "hello" + str(2) or "hello" + "2" to avoid the error.

🔤 Escape Characters

> **Escape character** – the backslash (\) that tells Python to treat the following character literally.

- Needed when a string contains the same quote character used to delimit it:

```
s = "Lucy\'s car is red"
```

- Prevents the apostrophe from terminating the string early.

📍 Indexing and Slicing

🔢 Index Basics

- Python strings are **zero-indexed**:
  - Index 0 → first character
  - Index 1 → second character
  - …
- Negative indices count from the end:
  - -1 → last character
  - -2 → second-last, etc.

📏 Positive Index Examples

```
a = "hello"
a[0]   # → 'h'
a[2]   # → 'l' (third character)
```

📐 Negative Index Examples

```
a[-1]  # → 'o'   (last character)
a[-2]  # → 'l'   (second-last)
a[-3]  # → 'l'   (third-last)
```

✂️ Slicing Syntax

> **Slice notation** – sequence[start:stop:step]
>
> - start – index to begin (inclusive)
> - stop – index to end (exclusive)

- step – jump between indices (default 1)

Basic Slicing

```
a[1:3]   # → 'el'   (indices 1 and 2)
a[1:4]   # → 'ell'  (indices 1,2,3)
a[3:5]   # → 'lo'   (indices 3,4)
```

Using a Step

```
a[0:5:2]   # → 'hlo'   (every second character)
a[::2]     # → 'hlo'   (whole string, step 2)
a[::-1]    # → 'olleh' (reverse the string)
```

Omitting Bounds

- a[:] → entire string.
- a[:3] → first three characters ('hel').
- a[2:] → from index 2 to the end ('llo').

Reverse Slicing

- a[::-1] → reversed string.
- a[3:-1] → from index 3 up to (but not including) the last character.

📚 Slicing Examples with Step

Slice	Result	Explanation
a[0:5]	hello	Full string (stop excluded at index 5)
a[1:4]	ell	Start at index 1, stop before index 4
a[::2]	hlo	Every second character
a[::-1]	olleh	Reverse order
a[1:5:2]	el	Indices 1 and 3
a[-3:]	llo	Last three characters

🛠️ Common String Operations

📏 len() Function

> **len()** – returns the number of characters in a string.

```
len("hello")   # → 5
```

🔄 Immutability

> **Immutable** – once a string is created, its characters cannot be changed individually.

- Attempting a[2] = "m" raises TypeError: 'str' object does not support item assignment.
- To “modify” a string, create a new one:

    ```
    a = "hello"
    a = a[:2] + "m" + a[3:]   # → "hemlo"
    ```

➕ Augmented Assignment with Strings

- += can concatenate and reassign in one step:

    ```
    a = "hello"
    a += " world"   # a → "hello world"
    ```