**Python Day 4: Slicing and Methods**

**1. Slicing: Extracting Subsequence’s**

Slicing is a powerful feature in Python that lets you extract a portion (a "slice") of a sequence. Instead of picking elements one by one with indexing, you can grab an entire chunk.

**The Slicing Syntax**

The basic syntax for slicing is sequence[start:stop:step].

* start: The index where the slice begins (this element **is included**).
* stop: The index where the slice ends (this element **is excluded**).
* step: The interval between elements to select. A step of 1 means every element, 2 means every other element, and so on. A negative step reverses the direction.

**Important Shorthands:**

* If start is omitted, it defaults to the beginning of the sequence (0).
* If stop is omitted, it defaults to the end of the sequence.
* If step is omitted, it defaults to 1.

**Positive and Negative Indexing Review**

Remember that sequences can be indexed from both ends:

# Positive Indexing (from the left, starts at 0)

P y t h o n

0 1 2 3 4 5

# Negative Indexing (from the right, starts at -1)

P y t h o n

-6 -5 -4 -3 -2 -1

**Slicing Strings**

Let's explore slicing using your examples. Our main string will be:

x = 'my name is omkar and i love python'

**Basic Slicing (Positive Step)**

This is the most common type of slicing, moving from left to right.

x = 'my name is omkar and i love python'

# Extract the first 10 characters

# Starts at index 0, stops BEFORE index 10, step is 1

z = x[0:10:1] # Can be written as x[:10]

print(z)

*Output:*

my name is

**Explanation:** The slice starts at index 0 ('m') and includes every character up to, but not including, the character at index 10 (' ', the space after 'is').

**1. Slicing (Continued)**

**Combining Slices**

You can perform multiple slices and concatenate (join) the results to rearrange a string.

**Example:**

Python

x = 'my name is omkar and i love python'

# Slice 1: 'my name is '

s1 = x[0:10:1]

# Slice 2: 'python'

s2 = x[-7:]

# Slice 3: ' and i love'

s3 = x[-18:-7:1]

# Slice 4: ' omkar'

s4 = x[10:16:1]

# Combine them in a new order

z = s1 + s2 + s3 + s4

print(z)

**Output:**

my name is python and i love omkar

**Explanation:** This shows how to deconstruct and reconstruct a string by taking specific "chunks" and joining them in a new order.

**Slicing with a Negative Step (Reversing)**

When the step is negative, the slice moves from right to left. The start index must be greater than the stop index.

**Example:**

x = 'my name is omkar and i love python'

# Slice from index 16 ('r') backwards to index 6 (' ')

# Start at 16, Stop before 6, Step is -1

z = x[16:6:-1]

print(z)

**Output:**

rakmo si

**Explanation:** The slice starts at x[16] ('r') and moves backward, stopping *before* reaching index 6.

**Common Pitfalls: Empty Slices**

A slice will be empty if the start, stop, and step values create an impossible request.

**Example:**

Python

x = 'my name is omkar and i love python'

# Attempt 1: Positive step can't go backwards

print(x[-1:-10:1])

# Attempt 2: Negative step can't go forwards

print(x[4:-1:-1])

**Output:**

(empty line)

(empty line)

**Slicing Lists and Tuples**

The exact same slicing principles apply to lists and tuples.

**Example:**

# 0 1 2 3 4 5 6 7 (Positive)

r = [ 1, 6, 7, 5, 4, 'R', 2, 6 ]

# -8 -7 -6 -5 -4 -3 -2 -1 (Negative)

# Slice 1: From index -3 ('R') backwards to index 2 (7)

s1 = r[-3:2:-1] # Result: ['R', 4, 5]

# Slice 2: From index 0 (1) forwards to index 3 (5)

s2 = r[0:3:1] # Result: [1, 6, 7]

v = s1 + s2

print(v)

**Output:**

['R', 4, 5, 1, 6, 7]

**2. Type Casting Errors and the NoneType**

Understanding why code fails is important.

**The Failing Code:**

# This code will fail

c = list(print(set(str([1,2,3,]))))

print(c)

**Output:**

{',', '3', '1', '2'}

TypeError: 'NoneType' object is not iterable

**Why the Error?**

1. The print() function's only job is to display a value on the screen.
2. After printing, it returns the special value None.
3. Your code then becomes c = list(None).
4. The list() function needs an "iterable" (like a string, set, or tuple) to build a list from. None is not iterable, so this fails with a TypeError.

The Correct Way to Code This:

To fix this, you should separate the operation from the printing.

Python

a = [1,2,3,]

s = set(str(a)) # Step 1: Perform the conversion and store in a variable

c = list(s) # Step 2: Perform the next conversion

print(c) # Step 3: Print the final result

**3. Methods: Built-in Actions for Data**

A **method** is a function that "belongs" to an object. You call it using dot notation (object.method()).

**Crucial Concept: Strings are Immutable**

String methods **do not change the original string**. They always return a **new, modified string**. To save the changes, you must assign the result back to a variable.

**Example:**

Python

a = 'i lOvE PyThon'

print("Original ID:", id(a))

# Calling .upper() returns a new string but 'a' is unchanged

a\_upper = a.upper()

print("New string:", a\_upper)

print("New string ID:", id(a\_upper))

print("Original string is still the same:", a)

**Output:**

Original ID: 139836042846640

New string: I LOVE PYTHON

New string ID: 139836042781552

Original string is still the same: i lOvE PyThon

**Explanation:** Notice the memory address (id) is different, proving a new string was created.

**The Core Principle: String Immutability**

A crucial concept is that strings are **immutable**. This means string methods **never change the original string**. Instead, they perform their operation and **return a new string** containing the result.

All examples use the string: a = 'i lOvE PyThon'

**Case Conversion Methods**

These methods are used to change the capitalization of the string.

* **a.lower()**
  + **Purpose:** Converts the entire string to lowercase.
  + **Result:** 'i love python'
* **a.upper()**
  + **Purpose:** Converts the entire string to uppercase.
  + **Result:** 'I LOVE PYTHON'
* **a.title()**
  + **Purpose:** Capitalizes the first letter of *each word* in the string.
  + **Result:** 'I Love Python'
* **a.capitalize()**
  + **Purpose:** Capitalizes *only the very first letter* of the entire string and converts the rest to lowercase.
  + **Result:** 'I love python'
* **a.swapcase()**
  + **Purpose:** Swaps the case of every letter (uppercase becomes lowercase and vice versa).
  + **Result:** 'I LoVe pYtHON'

**Substring Manipulation and Analysis**

These methods allow you to search, replace, and count parts of a string.

* **a.replace('old', 'new')**
  + **Purpose:** Replaces all occurrences of a specified substring (old) with another (new).
  + **Example:** a.replace('PyThon','java')
  + **Result:** 'i lOvE java'
* **a.count('substring')**
  + **Purpose:** Counts how many times a substring appears within the main string.
  + **Example:** a.count('O')
  + **Result:** 1
* **.find() vs. .index()**
  + **Purpose:** Both methods find the starting index of the *first* occurrence of a substring.
  + **Key Difference:** They handle failure differently.
    - **a.find('p')**: This is the "safe" method. If the substring is not found, it returns **-1**.
      * **Result:** -1
    - **a.index('P')**: This is the "strict" method. If the substring is not found, it raises a **ValueError**.
      * **Result:** 7

**General Purpose Function**

* **len(a)**
  + **Purpose:** This is a built-One-in **function**, not a method, used to get the length of an object.
  + **How It Works:** It returns the total number of characters in the string, including letters, symbols, and whitespace.
  + **Result:** 13