reversing

Python Reversing

Basic String Reversal

Method 1: Slicing (Most Pythonic)

```
def reverse_string_slice(s):
    return s[::-1]

# Example
text = "hello"
print(reverse_string_slice(text))  # Output: "olleh"
```

Method 2: Using reversed() and join()

```
def reverse_string_builtin(s):
    return ''.join(reversed(s))

# Example
text = "python"
print(reverse_string_builtin(text))  # Output: "nohtyp"
```

Method 3: Manual Loop (Interview Favorite)

```
def reverse_string_manual(s):
    result = ""
    for i in range(len(s) - 1, -1, -1):
        result += s[i]
    return result

# Alternative using while loop
def reverse_string_while(s):
    result = ""
    i = len(s) - 1
    while i ≥ 0:
        result += s[i]
        i -= 1
    return result
```

Method 4: Two-Pointer Technique (In-place)

```
def reverse_string_inplace(s):
    # Convert to list since strings are immutable
    s_list = list(s)
    left, right = 0, len(s_list) - 1

    while left < right:
        s_list[left], s_list[right] = s_list[right], s_list[left]
        left += 1
        right -= 1

    return ''.join(s_list)
```

Number Reversal

Method 1: String Conversion (Simple)

```
def reverse_number_string(num):
    # Handle negative numbers
    if num < 0:
        return -int(str(-num)[::-1])
    return int(str(num)[::-1])

# Examples
print(reverse_number_string(123))   # Output: 321
print(reverse_number_string(-456))  # Output: -654
```

Method 2: Mathematical Approach (Interview Preferred)

```
def reverse_number_math(num):
    # Handle negative numbers
    is_negative = num < 0
    num = abs(num)

    reversed_num = 0
    while num > 0:
        digit = num % 10
        reversed_num = reversed_num * 10 + digit
        num = num // 10

    return -reversed_num if is_negative else reversed_num

# Examples
```

```
print(reverse_number_math(12345))   # Output: 54321
print(reverse_number_math(-789))    # Output: -987
```

Handling Integer Overflow (LeetCode Style)

```
def reverse_with_overflow_check(x):
    INT_MAX = 2**31 - 1
    INT_MIN = -2**31

    is_negative = x < 0
    x = abs(x)

    result = 0
    while x:
        digit = x % 10

        # Check for overflow before updating result
        if result > INT_MAX // 10:
            return 0
        if result == INT_MAX // 10 and digit > INT_MAX % 10:
            return 0

        result = result * 10 + digit
        x //= 10

    result = -result if is_negative else result
    return result if INT_MIN ≤ result ≤ INT_MAX else 0
```

List/Array Reversal

Method 1: Slicing

```
def reverse_list_slice(arr):
    return arr[::-1]

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

Method 2: Built-in Methods

```
def reverse_list_builtin(arr):
    # Using reversed()
    return list(reversed(arr))
```

```python
def reverse_list_inplace(arr):
    # Modifies original list
    arr.reverse()
    return arr
```

Method 3: Two-Pointer Technique

```python
def reverse_list_two_pointer(arr):
    left, right = 0, len(arr) - 1

    while left < right:
        arr[left], arr[right] = arr[right], arr[left]
        left += 1
        right -= 1

    return arr
```

Advanced Reversing Scenarios

Reverse Words in a String

```python
def reverse_words(s):
    # Method 1: Using split and join
    return ' '.join(s.split()[::-1])

def reverse_words_manual(s):
    # Method 2: Manual approach
    words = []
    word = ""

    for char in s + " ":  # Add space to handle last word
        if char == " ":
            if word:
                words.append(word)
                word = ""
        else:
            word += char

    return ' '.join(words[::-1])

# Example
sentence = "Hello World Python"
print(reverse_words(sentence))  # Output: "Python World Hello"
```

Reverse Only Letters

```
def reverse_only_letters(s):
    s_list = list(s)
    left, right = 0, len(s_list) - 1

    while left < right:
        if not s_list[left].isalpha():
            left += 1
        elif not s_list[right].isalpha():
            right -= 1
        else:
            s_list[left], s_list[right] = s_list[right], s_list[left]
            left += 1
            right -= 1

    return ''.join(s_list)

# Example
text = "a-bC-dEf-ghIj"
print(reverse_only_letters(text))  # Output: "j-Ih-gfE-dCba"
```

Palindrome Check (Bonus)

```
def is_palindrome(s):
    # Clean the string: remove non-alphanumeric and convert to lowercase
    cleaned = ''.join(char.lower() for char in s if char.isalnum())
    return cleaned == cleaned[::-1]

def is_palindrome_two_pointer(s):
    left, right = 0, len(s) - 1

    while left < right:
        while left < right and not s[left].isalnum():
            left += 1
        while left < right and not s[right].isalnum():
            right -= 1

        if s[left].lower() != s[right].lower():
            return False

        left += 1
        right -= 1

    return True
```

Interview Practice Questions

Beginner Level

Question 1: Basic String Reversal

```
Write a function that takes a string and returns it reversed.
Input: "programming"
Output: "gnimmargorP"
```

Solution:

```python
def reverse_string(s):
    return s[::-1]
```

Question 2: Reverse a Number

```
Given an integer, reverse its digits.
Input: 12345
Output: 54321
```

Solution:

```python
def reverse_integer(x):
    is_negative = x < 0
    x = abs(x)

    result = 0
    while x:
        result = result * 10 + x % 10
        x //= 10

    return -result if is_negative else result
```

Intermediate Level

Question 3: Reverse Words in Sentence

```
Given a sentence, reverse the order of words.
Input: "The quick brown fox"
Output: "fox brown quick The"
```

Solution:

```python
def reverse_words(sentence):
    return ' '.join(sentence.split()[::-1])
```

Stop here for now

Question 4: Reverse Linked List

```
class ListNode:
    def __init__(self, val=0, next=None):
        self.val = val
        self.next = next

def reverse_linked_list(head):
    prev = None
    current = head

    while current:
        next_temp = current.next
        current.next = prev
        prev = current
        current = next_temp

    return prev
```

Advanced Level

Question 5: Reverse String with Constraints

```
Reverse a string but keep numbers in their original positions.
Input: "a1b2c3"
Output: "c1b2a3"
```

Solution:

```
def reverse_letters_only(s):
    s_list = list(s)
    left, right = 0, len(s_list) - 1

    while left < right:
        if not s_list[left].isalpha():
            left += 1
        elif not s_list[right].isalpha():
            right -= 1
        else:
            s_list[left], s_list[right] = s_list[right], s_list[left]
            left += 1
            right -= 1
```

```
    return ''.join(s_list)
```

Question 6: Scenario-Based Problem

```
You're building a text processing system for a chat application.
Users can type commands like "/reverse Hello World" and the system
should return "dlroW olleH". Handle edge cases like empty strings
and special characters.
```

Solution:

```
def chat_reverse_command(command):
    # Parse command
    if not command.startswith("/reverse "):
        return "Invalid command format"

    text_to_reverse = command[9:]  # Remove "/reverse "

    if not text_to_reverse:
        return "No text provided to reverse"

    return text_to_reverse[::-1]

# Test cases
print(chat_reverse_command("/reverse Hello World"))  # "dlroW olleH"
print(chat_reverse_command("/reverse "))  # "No text provided to reverse"
print(chat_reverse_command("reverse Hello"))  # "Invalid command format"
```

Real-World Scenarios

Scenario 1: Log File Analysis

```
def reverse_log_entries(log_file_path):
    """
    Reverse the order of entries in a log file for analysis.
    Most recent entries first.
    """
    try:
        with open(log_file_path, 'r') as file:
            lines = file.readlines()

        # Reverse and clean lines
        reversed_lines = [line.strip() for line in lines[::-1]]
        return reversed_lines
```

```
    except FileNotFoundError:
        return "Log file not found"
```

Scenario 2: URL Slug Reversal

```
def reverse_url_slug(url):
    """
    Reverse the components of a URL slug while maintaining structure.
    Input: "python-programming-basics"
    Output: "basics-programming-python"
    """
    if '/' in url:
        parts = url.split('/')
        slug = parts[-1]  # Get the last part
        base_url = '/'.join(parts[:-1])

        reversed_slug = '-'.join(slug.split('-')[::-1])
        return f"{base_url}/{reversed_slug}" if base_url else reversed_slug
    else:
        return '-'.join(url.split('-')[::-1])
```

Time & Space Complexity

String Reversal

- **Slicing (`s[::-1]`):** Time O(n), Space O(n)
- **Manual Loop:** Time O(n), Space O(n)
- **Two-Pointer:** Time O(n), Space O(n) [due to string immutability]

Number Reversal

- **String Method:** Time O(log n), Space O(log n)
- **Mathematical:** Time O(log n), Space O(1)

List Reversal

- **Slicing:** Time O(n), Space O(n)
- **In-place:** Time O(n), Space O(1)
- **Two-Pointer:** Time O(n), Space O(1)

Key Interview Tips

1. **Ask Clarifying Questions:**
   - Should I handle negative numbers?
   - What about integer overflow?
   - Are there any constraints on input size?
   - Should I modify the input or create a new output?
2. **Start Simple:**

- Begin with the most straightforward solution
- Optimize if asked or if you have time

3. **Edge Cases to Consider:**
   - Empty strings/arrays
   - Single character/element
   - Negative numbers
   - Very large numbers (overflow)
   - Special characters
   - Unicode characters
4. **Common Follow-ups:**
   - "Can you do it in-place?"
   - "What's the time/space complexity?"
   - "How would you handle very large inputs?"
   - "What if memory is limited?"
5. **Practice Variations:**
   - Reverse parts of a string/array
   - Reverse with conditions
   - Reverse data structures (linked lists, trees)
   - Reverse and validate (palindromes)

Remember: The key to mastering reversing problems is understanding the underlying patterns and being able to adapt them to different scenarios!