string qsns

January 30, 2026

[4]:
```
# 1.Given a string

# a.If the string contains all alphabets return the size of string

# b.If it contains all the digits then return square of the number

# c.else

# If first character is a digit then return the first and last character as a␣
↪list

# else find the longest common prefix for the string

def process_string(s):
    def  is_all_alphabets(s):
        return s.isalpha()
    def is_all_digits(s):
        return s.isdigit()
    def longest_common_prefix(s):
        if not s:
            return ""
        prefix=s[0]
        for i in range(len(prefix)):
            for string in s:
                if i>=len(string) or string[i]!=prefix[i]:
                    return prefix[:i]
        return prefix
    def first_and_last_char(s):
        if s[0].isdigit():
            return [s[0], s[-1]]
        return None

    if is_all_alphabets(s):
        return len(s)
    elif is_all_digits(s):
        return int(s)**2
    else:
        result = first_and_last_char(s)
        if result:
            return result
        else:
            return longest_common_prefix([s])


print(process_string("abcdef"))  # Output: 6
```

```
6
```

0.0.1 Advance String Questions

[5]:
```
# Write a function that cleans and normalizes text with rules:
# 1. If the string contains only digits → format as a phone number:␣
 ↪"1234567890" → "123-456-7890"
# 2. Elif string contains only alphabets:
# - If all uppercase → make lowercase
# - If all lowercase → make Title Case
# - Else → swapcase
# 3. Elif string contains spaces:
# - Collapse multiple spaces → single space
# - Trim leading/trailing spaces
# - Ensure the text ends with a period '.'
# 4. Else (mixed junk with symbols) → remove everything except alphanumeric and␣
 ↪spaces.

def clean_and_normalize_text(s):
    if s.isdigit():
        if len(s)==10:
            return s[:3]+'-'+s[3:6]+'-'+s[6:]
        return s

    elif s.isalpha():
        if s.isupper():
            return s.lower()
        elif s.islower():
            return s.title()
        else:
            return s.swapcase()

    elif ' ' in s:
        words=s.split() # removes extra spaces
        cleaned=' '.join(words).strip()
        if not cleaned.endswith('.'):
            cleaned+='.'
        return cleaned
    else:
        # mixed junk symbols
        cleaned=""
        for char in s:
            if char.isalnum() or char.isspace():
                cleaned+=char
        return cleaned


print(clean_and_normalize_text("   Hello   World   "))  # Output: "Hello World."
```

```
Hello World.
```

[6]:

```
# 2. Pattern-based String Transformer
# Given a string s, apply rules:
# 1. If string starts with "http":
# - If it also ends with ".org" → return "Organization URL"
# - Elif it ends with ".com" → return "Commercial URL"
# - Else → "Other URL"
# 2. Elif string contains "@":
# - If it ends with ".edu" → "Educational Email"
# - Else → "General Email"
# 3. Elif string is palindrome (ignore case & spaces) → "Palindrome String"
# 4. Else → "Unclassified"
# Test Cases:
# - "http://opensource.org" → "Organization URL"
# - "https://google.com" → "Commercial URL"
# - "user@mit.edu" → "Educational Email"
# - "user@gmail.com" → "General Email"
# - "A man a plan a canal Panama" → "Palindrome String"
# - "hello world" → "Unclassified"
def pattern_based_transformer(s):
    def isPalindrome(s):
        s=s.replace(" ","").lower()
        return s==s[::-1]
    if s.startswith("http"):
        if s.endswith(".org"):
            return "Organization URL"
        elif s.endswith(".com"):
            return "Commercial URL"
        else:
            return "Other URL"
    elif "@" in s:
        if s.endswith(".edu"):
            return "Educational Email"
        else:
            return "General Email"

    elif isPalindrome(s):
        return "Palindrome String"
    else:
        return "Unclassified"

print(pattern_based_transformer("A man a plan a canal Panama"))  # Output:␣
 ↪"Palindrome String"
```

```
Palindrome String
```

[7]:
```
# 3. Smart String Comparator
# Compare two strings a and b under rules:
# 1. If both strings are exactly equal → "Equal"
# 2. Elif their lowercase versions are equal → "Case-insensitive match"
# 3. Elif sorted characters of both strings are equal → "Anagrams"
# 4. Elif one string is substring of another → "Substring relation"
# 5. Else → "Completely different"
# Test Cases:
# - ("Hello", "Hello") → "Equal"
# - ("Hello", "hello") → "Case-insensitive match"
# - ("listen", "silent") → "Anagrams"
# - ("cat", "concatenate") → "Substring relation"
# - ("dog", "cat") → "Completely different"

def smart_string_comparator(a,b):
    if a==b:
        return "Equal"
    elif a.lower()==b.lower():
        return "Case-insensitive match"
    elif sorted(a)==sorted(b):
        return "Anagrams"
    elif a in b or b in a:
        return "Substring relation"
    else:
        return "Completely different"

print(smart_string_comparator("listen", "silent"))  # Output: "Anagrams"
```

```
Anagrams
```

[ ]:
```
# 4. Context-based String Masker
# Mask sensitive data with rules:
# 1. If string is 16-digit number → mask as credit card → "************5678"
# 2. Elif string looks like an email (contains @ and .):
# - Mask everything except first letter and domain → "j*****@gmail.com"
# 3. Elif string looks like URL (startswith http):
```

```
# - Keep domain only → "openai.com"
# 4. Else → "No sensitive data detected"
# Test Cases:
# - "1234567812345678" → "************5678"
# - "johndoe@gmail.com" → "j*****@gmail.com"
# - "https://platform.openai.com/account" → "openai.com"
# - "hello world" → "No sensitive data detected"

def context_based_string_masker(s):
    if s.isdigit() and len(s)==16:
        return "************"+s[-4:]
    elif "@" in s and "." in s:
        at_index=s.index("@")
        domain=s[at_index:]
        return s[0]+"*****"+domain
    elif s.startswith("http"):
        # extract domain
        s=s.replace("http://","")
        s=s.replace("https://","")
        domain=s.split("/")[0]
        # domain=".".join(domain.split(".")[-2:])
        return domain
    else:
        return "No sensitive data detected"
```

[8]:
```
# 5. Complex Password Auditor
# Audit password and return a detailed category:
# 1. If length < 6 → "Invalid: Too Short"
# 2. Elif password is all digits → "Invalid: Only Numbers"
# 3. Elif password is all letters:
# - If all lowercase → "Weak: Only Lowercase Letters"
# - If all uppercase → "Weak: Only Uppercase Letters"
# - Else → "Weak: Only Letters"
# 4. Elif password has digits and letters but no special chars → "Moderate:␣
 ↪Needs Special Char"
# 5. Elif password has digits, letters, and special chars but no uppercase →␣
 ↪"Strong but Missing
# Uppercase"
# 6. Else → "Very Strong"
# Test Cases:
# - "12345" → "Invalid: Too Short"
# - "123456" → "Invalid: Only Numbers"
# - "abcdef" → "Weak: Only Lowercase Letters"
# - "ABCDEF" → "Weak: Only Uppercase Letters"
# - "abcDEF" → "Weak: Only Letters"
# - "abc123" → "Moderate: Needs Special Char"
# - "abc123@" → "Strong but Missing Uppercase"
# - "Abc123@" → "Very Strong

def complex_password_auditor(password):
    special_chars="!@#$%^&*()-_=+[]{}|;:'\",.<>?/`~"
    if len(password)<6:
        return "Invalid: Too Short"
    elif password.isdigit():
        return "Invalid: Only Numbers"
    elif password.isalpha():
        if password.islower():
            return "Weak: Only Lowercase Letters"
        elif password.isupper():
            return "Weak: Only Uppercase Letters"
        else:
            return "Weak: Only Letters"
    else:
        has_digit=any(char.isdigit() for char in password)
        has_letter=any(char.isalpha() for char in password)
        has_special=any(char in special_chars for char in password)
        has_upper=any(char.isupper() for char in password)

        if has_digit and has_letter and not has_special:
            return "Moderate: Needs Special Char"
        elif has_digit and has_letter and has_special and not has_upper:
            return "Strong but Missing Uppercase"
        else:
            return "Very Strong"

print(complex_password_auditor("abc123@"))  # Output: "Strong but Missing␣
 ↪Uppercase"
```

```
Strong but Missing Uppercase
```

```
[ ]:
```

```
[ ]:
```