System Requirements Specification Index

For

Digital Communication Analyzer

Version 1.0

IIHT Pvt. Ltd.
fullstack@iiht.com

TABLE OF CONTENTS

1 PROJECT ABSTRACT

The Digital Communications Team requires a simple text analysis system to identify and extract common patterns within text data. This assignment focuses on implementing basic regular expression functions to match, extract, and validate common data formats like emails, phone numbers, and dates. Students will learn regex fundamentals while building practical pattern-matching utilities.

2 BUSINESS REQUIREMENTS:

Screen Name	Console input screen
Problem Statement	1. System must implement basic regex pattern matching functions 2. Each function must handle a specific pattern type (emails, phone numbers, etc.) 3. Functions must include proper documentation (docstrings) 4. System must validate and extract common data formats from text 5. All functions must handle edge cases and provide appropriate error handling

3 CONSTRAINTS

3.1 INPUT REQUIREMENTS

1. Text Data Format:
    - Text data must be provided as strings

- Example: `"Please contact john.doe@example.com or call (555) 123-4567"`

2. Pattern Types:
    - Email addresses: Standard format email addresses including special characters
    - Phone numbers: Common US phone number formats (parentheses, dashes, dots, spaces)
    - Dates: Common date formats (MM/DD/YYYY, YYYY-MM-DD) with single or double digits
    - IP addresses: IPv4 format with validation for valid octet ranges (0-255)
3. Edge Cases:
    - Empty inputs should return empty lists for extraction functions
    - Inputs with no matches should return empty lists
    - Invalid inputs (non-strings) should raise appropriate exceptions

3.2 Function Constraints

1. Function Definition:
    - Each function must have a clear purpose
    - Must include docstrings with examples
    - Example: `def extract_emails(text):`
2. Regular Expression Patterns:
    - Must use the Python `re` module
    - Must use raw strings for patterns (r"pattern")
    - Must handle common format variations
    - Patterns must be flexible enough to handle special characters where appropriate
3. Return Values:
    - Functions must return consistent data types
    - Extraction functions must return lists of matches
    - Validation functions must return boolean values
    - All extraction functions must handle text without matches by returning empty lists
4. Error Handling:
    - All functions must verify input types
    - Raise TypeError for non-string inputs
    - Extraction functions should not raise exceptions for valid inputs with no matches

- Date extraction should validate format parameter and raise ValueError for invalid formats

3.3 Implementation Constraints

1. Email Pattern Details:
    - Username part: Letters, numbers, dots, underscores, percent, plus, and hyphen characters
    - Domain part: Letters, numbers, dots, and hyphens
    - TLD: Two or more letters
    - Example regex: `r'[a-zA-Z0-9._%+-]+@[a-zA-Z0-9.-]+\.[a-zA-Z]{2,}'`
2. Phone Number Pattern Details:
    - Area code: May be in parentheses or not
    - Separators: May use dashes, dots, spaces, or none
    - Format: 3 digits (area code) + 3 digits + 4 digits
    - Example regex: `r'\(?\d{3}\)?[-.\s]?\d{3}[-.\s]?\d{4}'`
3. Date Pattern Details:
    - MM/DD/YYYY: Month and day may be 1 or 2 digits, year is 4 digits
    - YYYY-MM-DD: Year is 4 digits, month and day may be 1 or 2 digits
    - Example US format: `r'\b\d{1,2}/\d{1,2}/\d{4}\b'`
    - Example ISO format: `r'\b\d{4}-\d{1,2}-\d{1,2}\b'`
4. IP Address Pattern Details:
    - Four octets of 1-3 digits each, separated by dots
    - Each octet must be validated to ensure it's in range 0-255
    - Example regex for matching: `r'\b\d{1,3}\.\d{1,3}\.\d{1,3}\.\d{1,3}\b'`
    - Additional validation needed for octet range check
5. Pattern Replacement Details:
    - Use `re.sub()` to replace all occurrences of a pattern
    - Ensure replacement affects all matches, not just the first one

3.4 Output Constraints

1. Display Format:
    - Console output must be clearly labeled and organized

- Example: `Found 3 email addresses: user@example.com, admin@test.org, info@company.net`

4. TEMPLATE CODE STRUCTURE:

1. Pattern Extraction Functions:

- `extract_emails(text)` - extracts email addresses
  - Must handle various email formats including special characters, subdomains
  - Return all matches as a list
- `extract_phone_numbers(text)` - extracts phone numbers
  - Must handle formats like (123) 456-7890, 123-456-7890, 123.456.7890, 123 456 7890
  - Return all matches as a list
- `extract_dates(text, format)` - extracts dates in specified format
  - Support "MM/DD/YYYY" and "YYYY-MM-DD" formats
  - Handle single-digit and double-digit month/day formats
  - Return all matches as a list
- `extract_ip_addresses(text)` - extracts IPv4 addresses
  - Match standard format IPv4 addresses
  - Return all matches as a list

2. Validation Functions:

- `validate_email(email)` - validates a single email address
  - Must check for proper format with username, @ symbol, and domain
  - Return boolean result
- `validate_phone_number(phone)` - validates a single phone number
  - Must handle multiple formats (parentheses, dashes, dots, spaces)
  - Return boolean result
- `validate_ip_address(ip)` - validates an IPv4 address
  - Check format and validate each octet is in range 0-255
  - Return boolean result

3. Pattern Replacement Function:

- `replace_pattern(text, pattern, replacement)` - replaces all pattern matches
  - Replace ALL occurrences, not just the first one
  - Return modified text

4. Main Program Function:

- `main()` - demonstrates all other functions
  - Include sample text with various patterns

- Show extraction and validation of each pattern type
- Demonstrate pattern replacement

5. EXECUTION STEPS TO FOLLOW:

1. Run the program
2. Observe the execution of each regex function
3. Test extraction and validation on sample text
4. Test with different input formats to verify flexibility
5. Verify error handling with invalid inputs
6. Test boundary conditions (empty strings, edge cases)
7. Validate pattern replacement with multiple occurrences