1. Write a program that takes the user's name and pan card number as input. Validate the information using isX function and print the details.

```
def isValidName(name):
    """
    Validate if the name contains only alphabets and spaces.

    Args:
    name (str): Name to validate

    Returns:
    bool: True if name is valid, False otherwise
    """
    return name.replace(" ", "").isalpha() and len(name.strip()) > 0

def isValidPAN(pan):
    """
    Validate PAN card number format.
    Rules:
    - Must be 10 characters long
    - First 5 characters must be uppercase letters
    - Next 4 characters must be numbers
    - Last character must be an uppercase letter

    Args:
    pan (str): PAN number to validate

    Returns:
    bool: True if PAN is valid, False otherwise
    """
    # Convert to uppercase for validation
    pan = pan.upper()

    # Check length
    if len(pan) != 10:
        return False

    # Check first 5 characters are letters
    if not pan[:5].isalpha():
        return False
    # Check next 4 characters are digits
    if not pan[5:9].isdigit():
        return False

    # Check last character is letter
    if not pan[9].isalpha():
        return False

    return True

def get_validated_input(prompt, validator_func, error_message):
    """
    Get input from user and validate it using the provided validator function.

    Args:
    prompt (str): Input prompt for the user
    validator_func (function): Function to validate the input
    error_message (str): Message to display if validation fails

    Returns:
    str: Validated input
    """
    while True:
        user_input = input(prompt).strip()
        if validator_func(user_input):
            return user_input
        print(error_message)

# Main program
print("PAN Card Details Validator")
print("-" * 25)

# Get and validate name
name = get_validated_input(
    "Enter your name: ",
    isValidName,
    "Invalid name! Name should contain only alphabets and spaces."
)

# Get and validate PAN number
pan = get_validated_input(
    "Enter your PAN card number: ",
    isValidPAN,
    "Invalid PAN number! PAN should be in format: AAAAA9999A"
)
```

```
    # Print validated details
    print("\nValidated Details:")
    print("-" * 25)
    print(f"Name: {name}")
    print(f"PAN Number: {pan.upper()}")
```

```
PAN Card Details Validator
-------------------------
Enter your name:  Ashadullah danish
Enter your PAN card number:  eJIPD58QWQ
Invalid PAN number! PAN should be in format: AAAAA9999A
Enter your PAN card number:  EJIPD5158Z

Validated Details:
-------------------------
Name: Ashadullah danish
PAN Number: EJIPD5158Z
```

2. Write a program to generate an Abecedarian series. (a series in which elements appears in an alphabetical order)

```
def is_abecedarian(word):
    """
    Check if a word is abecedarian (letters in alphabetical order).

    Args:
    word (str): Word to check

    Returns:
    bool: True if word is abecedarian, False otherwise
    """
    # Convert to lowercase for comparison
    word = word.lower()
    # Compare with sorted version of the word
    return word == ''.join(sorted(word))

def generate_abecedarian_words(length):
    """
    Generate abecedarian words of given length.

    Args:
    length (int): Length of words to generate

    Returns:
        list: List of abecedarian words
        """
        import string
        def generate_recursive(prev_chars, remaining_length, current_pos):
            # Base case: if we've reached desired length, return the word
            if remaining_length == 0:
                return [prev_chars]

            result = []
            # Start from the position after the last character used
            for i in range(current_pos, 26-(remaining_length-1)):
                # Add new character and recursively generate rest
                new_word = prev_chars + string.ascii_lowercase[i]
                result.extend(generate_recursive(new_word, remaining_length-1, i+1))
            return result

        return generate_recursive("", length, 0)

    def print_series(words):
        """
        Print the series in a formatted way.
        """
        for i, word in enumerate(words, 1):
            print(f"{i}. {word}")

    # Main program
    print("Abecedarian Series Generator")
    print("-" * 30)

    # Generate series of different lengths
    print("\nAbecedarian series of length 2:")
    series2 = generate_abecedarian_words(2)
    print_series(series2)

    print("\nAbecedarian series of length 3:")
    series3 = generate_abecedarian_words(3)
    print_series(series3)

    print("\nAbecedarian series of length 4:")
    series4 = generate_abecedarian_words(4)
    print_series(series4)

    # Example of checking if specific words are abecedarian
    print("\nChecking specific words:")
```

```
test_words = ["act", "dog", "flow", "agile", "below"]
for word in test_words:
    if is_abecedarian(word):
        print(f"'{word}' is an abecedarian word")
    else:
        print(f"'{word}' is not an abecedarian word")

# Allow user to check their own words
print("\nCheck your own words:")
while True:
    word = input("\nEnter a word to check (or press Enter to quit): ")
    if not word:
        break
    if is_abecedarian(word):
        print(f"'{word}' is an abecedarian word")
    else:
        print(f"'{word}' is not an abecedarian word")
```

```
Abecedarian Series Generator
------------------------------

Abecedarian series of length 2:
1. ab
2. ac
3. ad
4. ae
5. af
6. ag
7. ah
8. ai
9. aj
10. ak
11. al
12. am
13. an
14. ao
15. ap
16. aq
17. ar
18. as
19. at
20. au
21. av
22. aw
23. ax
24. ay
25. az
26. bc
27. bd
28. be
29. bf
30. bg
31. bh
32. bi
33. bj
```

```
34. bk
35. bl
36. bm
37. bn
38. bo
39. bp
40. bq
41. br
42. bs
43. bt
44. bu
45. bv
46. bw
47. bx
48. by
49. bz
50. cd
51. ce
52. cf
53. cg
54. ch
```

3. Write a program that counts the occurrences of a character in a string. Do not use built in functions. 3.

```
def count_character(string, char):
    """
    Count occurrences of a character in a string without using built-in functions.

    Args:
        string: The input string to search through
        char: The character to count

    Returns:
        count: Number of times the character appears in the string
    """
    if not string or not char:
        return 0

    count = 0
    index = 0

    # Manually iterate through each character
    while index < len(string):
        if string[index] == char:
            count = count + 1
        index = index + 1
    return count

# Test cases
def run_tests():
    test_cases = [
        ("hello world", "l", 3),
        ("", "a", 0),
        ("aaa", "a", 3),
        ("testing", "z", 0),
        ("Hello World", "o", 2)
    ]

    for test_string, test_char, expected in test_cases:
        result = count_character(test_string, test_char)
        print(f"String: {test_string}")
        print(f"Character: {test_char}")
        print(f"Expected: {expected}")
        print(f"Got: {result}")
        print(f"Pass: {result == expected}\n")

# Run the tests
run_tests()
```

```
String: hello world
Character: l
Expected: 3
Got: 3
Pass: True

String: 
Character: a
Expected: 0
Got: 0
Pass: True

String: aaa
Character: a
Expected: 3
Got: 3
Pass: True

String: testing
Character: z
Expected: 0
Got: 0
Pass: True

String: Hello World
Character: o
Expected: 2
Got: 2
Pass: True
```

4. Write a function that takes a list of words and returns the length of the longest one.

```
def get_string_length(string):
    """
    Calculate length of string without using built-in len() function

    Args:
        string: Input string to measure

    Returns:
        length: Number of characters in the string
    """
    count = 0
    for _ in string:
        count += 1
    return count

def find_longest_word_length(words):
    """
    Find the length of the longest word in a list of words

    Args:
        words: List of strings to analyze

    Returns:
        max_length: Length of the longest word found
    """
    if not words:
        return 0

    max_length = 0

    for word in words:
        current_length = get_string_length(word)
        if current_length > max_length:
            max_length = current_length

    return max_length

# Test cases
def run_tests():
    test_cases = [
        (["hello", "world", "python", "programming"], 11),  # "programming" is longest
        (["a", "ab", "abc"], 3),  # "abc" is longest
        ([], 0),  # empty list
        (["x"], 1),  # single character
        (["", "test", ""], 4)  # includes empty strings
    ]

    for test_words, expected in test_cases:
        result = find_longest_word_length(test_words)
        print(f"Words: {test_words}")
        print(f"Expected: {expected}")
        print(f"Got: {result}")
        print(f"Pass: {result == expected}\n")

# Run the tests
run_tests()
```

```
Words: ['hello', 'world', 'python', 'programming']
Expected: 11
Got: 11
Pass: True

Words: ['a', 'ab', 'abc']
Expected: 3
Got: 3
Pass: True

Words: []
Expected: 0
Got: 0
Pass: True

Words: ['x']
Expected: 1
Got: 1
Pass: True

Words: ['', 'test', '']
Expected: 4
Got: 4
Pass: True
```

5. Write a function to get the first half of half of a specified string of even length.

```
def get_first_quarter(string):
    """
    Get the first quarter (half of half) of a string that must have length divisible by 4
```

```
        Args:
            string: Input string with length divisible by 4

        Returns:
            The first quarter of the input string
            Raises ValueError if string length is not divisible by 4
        """
        # Calculate length without using len()
        length = 0
        for _ in string:
            length += 1

        # Check if length is divisible by 4
        if length % 4 != 0:
            raise ValueError("String length must be divisible by 4")

        # Calculate quarter point
        quarter_point = length // 4

        # Build first quarter without using string slicing
        result = ""
        index = 0
        while index < quarter_point:
            result += string[index]
            index += 1

        return result

    # Test cases
    def run_tests():
        test_cases = [
            ("programming", False),  # length 11, not divisible by 4
            ("pythoncode", False),  # length 10, not divisible by 4
            ("teststring", True),   # length 10, returns "test"
            ("abcd", True),         # length 4, returns "a"
            ("", True),             # length 0, returns ""
            ("HelloWorld!", False), # length 11, not divisible by 4
            ("abcdefgh", True)      # length 8, returns "ab"
        ]

        for test_string, should_succeed in test_cases:
            print(f"Testing string: '{test_string}'")
            try:
```

```
            result = get_first_quarter(test_string)
            print(f"Result: '{result}'")
            print(f"Pass: {should_succeed}\n")
        except ValueError as e:
            print(f"Error: {e}")
            print(f"Pass: {not should_succeed}\n")

# Run the tests
run_tests()
```

```
Testing string: 'programming'
Error: String length must be divisible by 4
Pass: True

Testing string: 'pythoncode'
Error: String length must be divisible by 4
Pass: True

Testing string: 'teststring'
Error: String length must be divisible by 4
Pass: False

Testing string: 'abcd'
Result: 'a'
Pass: True

Testing string: ''
Result: ''
Pass: True

Testing string: 'HelloWorld!'
Error: String length must be divisible by 4
Pass: True

Testing string: 'abcdefgh'
Result: 'ab'
Pass: True
```

6. Write a program to get a single string from two given strings separated by a space and swap the first two characters of each string.

```
def swap_first_two_chars(string1, string2):
    """
    Takes two strings, swaps their first two characters, and combines them with a space

    Args:
        string1: First input string (must be at least 2 characters)
        string2: Second input string (must be at least 2 characters)
        Returns:
            Combined string with first two characters of each input swapped
            Raises ValueError if either string is less than 2 characters
        """
        # Verify string lengths without using len()
        length1 = 0
        length2 = 0

        for _ in string1:
            length1 += 1
        for _ in string2:
            length2 += 1

        if length1 < 2 or length2 < 2:
            raise ValueError("Both strings must be at least 2 characters long")

        # Build new strings with swapped characters
        new_string1 = ""
        new_string2 = ""

        # First two chars of string2 + rest of string1
        new_string1 += string2[0]
        new_string1 += string2[1]

        index = 2
        while index < length1:
            new_string1 += string1[index]
            index += 1

        # First two chars of string1 + rest of string2
        new_string2 += string1[0]
        new_string2 += string1[1]

        index = 2
        while index < length2:
            new_string2 += string2[index]
            index += 1

        # Return combined string with space
        return new_string1 + " " + new_string2

    # Test cases
    def run_tests():
```

```
    test_cases = [
        ("hello", "world", "wollo herld"),   # Basic case
        ("python", "java", "jathon pyva"),   # Different length strings
        ("ab", "cd", "cd ab"),               # Minimum length strings
        ("a", "world", None),                # First string too short
        ("hello", "w", None),                # Second string too short
        ("coding", "time", "ticoding code"), # Same length strings
        ("", "", None)                       # Empty strings
    ]

    for str1, str2, expected in test_cases:
        print(f"\nTesting strings: '{str1}' and '{str2}'")
        try:
            result = swap_first_two_chars(str1, str2)
            print(f"Result: '{result}'")
            print(f"Expected: '{expected}'")
            print(f"Pass: {result == expected}")
        except ValueError as e:
            print(f"Error: {e}")
            print(f"Pass: {expected is None}")

# Run the tests
run_tests()
```

```

Testing strings: 'hello' and 'world'
Result: 'wollo herld'
Expected: 'wollo herld'
Pass: True

Testing strings: 'python' and 'java'
Result: 'jathon pyva'
Expected: 'jathon pyva'
Pass: True

Testing strings: 'ab' and 'cd'
Result: 'cd ab'
Expected: 'cd ab'
Pass: True

Testing strings: 'a' and 'world'
Error: Both strings must be at least 2 characters long
Pass: True

Testing strings: 'hello' and 'w'
Error: Both strings must be at least 2 characters long
Pass: True

Testing strings: 'coding' and 'time'
Result: 'tiding come'
Expected: 'ticoding code'
Pass: False
```

```
Testing strings: '' and ''
Error: Both strings must be at least 2 characters long
Pass: True
```

7. Write a program to print floating point numbers with no decimal places.

```
def remove_decimal_places(number):
    """
    Convert a floating point number to an integer by removing decimal places

    Args:
        number: Input floating point number

    Returns:
        Integer part of the number as a string
    """
    # Convert number to string without using str()
    if number < 0:
        is_negative = True
        number = -number
    else:
        is_negative = False

    # Build integer part character by character
    result = ""
    number_str = f"{number}"  # Using f-string as we need string representation

    for char in number_str:
        if char == '.':
            break
        result += char

    # Handle empty result (case of numbers like 0.5)
    if result == "":
        result = "0"

    # Add negative sign if needed
    if is_negative:
        result = "-" + result

    return result
```

```
def print_without_decimals(numbers):
    """
    Print a list of floating point numbers without decimal places

    Args:
        numbers: List of floating point numbers
    """
    for number in numbers:
        result = remove_decimal_places(number)
        print(f"Original: {number}, Without decimals: {result}")

# Test cases
def run_tests():
    test_cases = [
        [123.45, 0.89, 9.0, -5.123, 0.23, -0.589, 1000.2],
        [0.5, -0.7, 0.0],
        [1.0, 2.0, 3.0],
        [-1.23, -0.0, 0.0]
    ]

    for test_set in test_cases:
        print("\nTesting set:", test_set)
        print_without_decimals(test_set)
        print("-" * 50)

# Run the tests
run_tests()
```

```

Testing set: [123.45, 0.89, 9.0, -5.123, 0.23, -0.589, 1000.2]
Original: 123.45, Without decimals: 123
Original: 0.89, Without decimals: 0
Original: 9.0, Without decimals: 9
Original: -5.123, Without decimals: -5
Original: 0.23, Without decimals: 0
Original: -0.589, Without decimals: -0
Original: 1000.2, Without decimals: 1000
--------------------------------------------------

Testing set: [0.5, -0.7, 0.0]
Original: 0.5, Without decimals: 0
Original: -0.7, Without decimals: -0
Original: 0.0, Without decimals: 0
--------------------------------------------------

Testing set: [1.0, 2.0, 3.0]
Original: 1.0, Without decimals: 1
Original: 2.0, Without decimals: 2
Original: 3.0, Without decimals: 3
--------------------------------------------------
```

```
Testing set: [-1.23, -0.0, 0.0]
Original: -1.23, Without decimals: -1
Original: -0.0, Without decimals: -0
Original: 0.0, Without decimals: 0
--------------------------------------------------
```

Start coding or generate with AI.