Aim-

write a python program that contains a string(char pointer) with a value \Hello world. The program should AND or and XOR each character in this string with 127 and display the result.

Program-

```
def and_xor_operations(input_string):
    print(f"Original string: {input_string}")
    print("\nResults:")
    print(f"{'Character':<10}{'AND with 127':<15}{'XOR with 127':<15}")
    print("-" * 40)

    for char in input_string:
        and_result = ord(char) & 127  # Perform AND operation with 127
        xor_result = ord(char) ^ 127  # Perform XOR operation with 127
        print(f"{char:<10}{and_result:<15}{xor_result:<15}")

# Input string
input_string = "\\Hello world"
and_xor_operations(input_string)
```

Explaination-

Sure! Here's a detailed line-by-line explanation of the Python program:

Code:

```
def and_xor_operations(input_string):
```

- **Purpose**: Define a function `and_xor_operations` that takes one parameter, `input_string`.
- **Input**: A string whose characters will be processed using AND and XOR operations with the value `127`.

```
print(f"Original string: {input_string}")
```

- **Purpose**: Print the original string for reference.
- **`f-string`**: Formats the string to include the value of `input_string`.

```
print("\nResults:")
```

- **Purpose**: Add a newline (`\n`) for better formatting, followed by a heading, `"Results:"`.

```
print(f"{'Character':<10}{'AND with 127':<15}{'XOR with 127':<15}")
```

- **Purpose**: Print column headers: `"Character"`, `"AND with 127"`, and `"XOR with 127"`.
- **`:<10`** and **`:<15`**: Format the output to align columns to a width of 10 or 15 spaces, creating a clean table layout.

```
print("-" * 40)
```

- **Purpose**: Print a separator line of 40 dashes (`-`) for readability.

```
for char in input_string:
```

- **Purpose**: Iterate through each character of the `input_string`.
- **`char`**: Represents the current character in the loop.

```
    and_result = ord(char) & 127
```

- **Purpose**: Compute the result of a bitwise AND operation between the ASCII value of `char` and `127`.
- **`ord(char)`**: Converts the character to its ASCII value.
- **`& 127`**: Performs a bitwise AND operation between the ASCII value and `127`.

```
    xor_result = ord(char) ^ 127
```

- **Purpose**: Compute the result of a bitwise XOR operation between the ASCII value of `char` and `127`.
- **`^ 127`**: Performs a bitwise XOR operation between the ASCII value and `127`.

```
    print(f"{char:<10}{and_result:<15}{xor_result:<15}")
```

- **Purpose**: Print the current character (`char`), its AND result (`and_result`), and its XOR result (`xor_result`).
- **`:<10`** and **`:<15`**: Align the outputs in table columns, ensuring consistent formatting.

```
# Input string
input_string = "\\Hello world"
```

- **Purpose**: Define the input string `\Hello world` to be processed.
- **Note**: The `\` character is treated as a regular character because it's written as `\\` (escaped backslash).

```
and_xor_operations(input_string)
```

- **Purpose**: Call the `and_xor_operations` function with `input_string` as an argument to perform the AND and XOR operations and display the results.

Key Concepts Explained:

1. **ASCII Conversion (`ord()`):** Converts characters to their corresponding ASCII values.
    - Example: `ord('H')` returns `72`.
2. **Bitwise AND (`&`):** Compares each bit of two numbers. If both bits are `1`, the result is `1`; otherwise, it's `0`.
    - Example: `72 & 127` results in `72`.
3. **Bitwise XOR (`^`):** Compares each bit of two numbers. If the bits are different, the result is `1`; if the bits are the same, the result is `0`.
    - Example: `72 ^ 127` results in `55`.
4. **Formatting:** Using `f-strings` for formatted output and alignment ensures a neat and readable table.

How It Works:

1. The input string `\Hello world` is processed character by character.
2. For each character, its ASCII value undergoes two operations:
    - AND with `127`.
    - XOR with `127`.
3. The results, along with the character, are printed in a table.

This program illustrates how bitwise operations affect ASCII values and how Python's formatting tools can be used for clean output.