Functions

1. **Calculate Factorial**

- **Problem Statement**: Write a function `factorial(n)` that takes a positive integer `n` and returns its factorial. The factorial of a number `n` is the product of all positive integers less than or equal to `n`.
- **Example**:

```
factorial(5)  # Output: 120
```

2. **Check Palindrome**

- **Problem Statement**: Create a function `is_palindrome(s)` that checks whether a given string `s` is a palindrome. A palindrome is a word that reads the same forwards and backwards.
- **Example**:

```
is_palindrome("radar")  # Output: True
is_palindrome("hello")  # Output: False
```

3. **Fibonacci Sequence Generator**

- **Problem Statement**: Write a function `fibonacci(n)` that returns the first `n` numbers of the Fibonacci sequence.
- **Example**:

```
fibonacci(5)  # Output: [0, 1, 1, 2, 3]
```

4. **Count Vowels in a String**

- **Problem Statement**: Implement a function `count_vowels(s)` that counts the number of vowels (`a, e, i, o, u`) in a given string `s`.
- **Example**:

```
count_vowels("hello world")  # Output: 3
```

5. **Prime Number Checker**

- **Problem Statement**: Create a function `is_prime(n)` that checks if a given number `n` is prime.
- **Example**:

```
is_prime(11)  # Output: True
is_prime(10)  # Output: False
```

6. **Flatten a List of Lists**

- **Problem Statement**: Write a function `flatten_list(nested_list)` that takes a list of lists and flattens it into a single list.
- **Example**:

```
flatten_list([[1, 2], [3, 4], [5]])  # Output: [1, 2, 3, 4, 5]
```

7. **Calculate GCD of Two Numbers**

- **Problem Statement**: Implement a function `gcd(a, b)` that finds the greatest common divisor of two numbers using the Euclidean algorithm.
- **Example**:

```
gcd(48, 18)  # Output: 6
```

8. **Generate Pascal's Triangle**

- **Problem Statement**: Write a function `pascals_triangle(n)` that generates `n` rows of Pascal's Triangle.
- **Example**:

```
pascals_triangle(5)
# Output:
# [
#   [1],
#   [1, 1],
#   [1, 2, 1],
#   [1, 3, 3, 1],
#   [1, 4, 6, 4, 1]
# ]
```

9. **Complex Problem 1: Library Management System**

- **Problem Statement**: Create a mini library management system using functions.
- **Requirements**:
  - Create a function `add_book(library, book_name, author)` to add a new book to the library.
  - Create a function `search_books(library, query)` to search for books by name or author.
  - Create a function `borrow_book(library, book_name)` to mark a book as borrowed.
  - Create a function `return_book(library, book_name)` to return a borrowed book.

- **Example**:

```
library = []
add_book(library, "Python 101", "John Doe")
add_book(library, "Data Science Handbook", "Jane Doe")
search_books(library, "Python")  # Output: [{"name": "Python 101", "author":
"John Doe", "status": "available"}]
borrow_book(library, "Python 101")
return_book(library, "Python 101")
```

10. **Complex Problem 2: Tic-Tac-Toe Game Implementation**

- **Problem Statement**: Create a complete implementation of a Tic-Tac-Toe game using functions.
- **Requirements**:
  - A function `print_board(board)` to print the current board state.
  - A function `check_winner(board)` to check if there is a winner or the game is a draw.
  - A function `make_move(board, player, position)` to make a move for a player (`X` or `O`).
  - A function `play_game()` that manages the game loop until there's a winner or a draw.
- **Example**:

```
play_game()
# Output:
# Player X, enter your move (1-9): 5
# Current board:
# [1, 2, 3]
# [4, X, 6]
# [7, 8, 9]
# ...
# Player O wins!
```