Pseudo Code Cheat Sheet
This cheat sheet covers the essential concepts and syntax to help you prepare for assessments that involve writing or analyzing pseudo-code.

Basic Structure of Pseudo Code
Pseudo-code is a plain-language way to describe algorithms and processes without strict syntax. The goal is to focus on **logic** rather than implementation.
Example Structure:

```
BEGIN
   Read Input
   Process Data
   Output Result
END
```

Key Components in Pseudo Code

1. **Variables and Assignment**
    - Used to store and manipulate data.
    - Example:

```
x = 5
y = 10
sum = x + y
```

2. **Conditional Statements (IF-ELSE)**
    - Controls the flow of the algorithm based on conditions.
    - Example:

```
IF x > y THEN
   PRINT "x is greater"
ELSE
   PRINT "y is greater"
END IF
```

3. **Loops (FOR, WHILE)**
    - Used to repeat a set of instructions multiple times.
    - **FOR Loop**:

```
FOR i = 1 TO 5 DO
   PRINT i
END FOR
```

    - **WHILE Loop**:

```
WHILE x > 0 DO
   PRINT x
   x = x - 1
END WHILE
```

4. **Functions/Procedures**
    - A block of code designed to perform a specific task.
    - Example:

```
FUNCTION add(x, y)
   RETURN x + y
```

```
END FUNCTION
```

Recursion

Recursion is a key concept in pseudo-code. A recursive function is a function that calls itself to solve a smaller instance of the same problem.

Key Points:

- **Base Case**: The condition where the recursion stops.
- **Recursive Call**: The part of the function that calls itself with modified input.

Example (Factorial):

```
FUNCTION factorial(n)
  IF n == 0 THEN
    RETURN 1
  ELSE
    RETURN n * factorial(n - 1)
  END IF
END FUNCTION
```

- In the above example, the base case is n == 0, and the recursive call is factorial(n - 1).

Common Patterns

Summation Using Recursion:

```
FUNCTION sum(n)
  IF n == 1 THEN
    RETURN 1
  ELSE
    RETURN n + sum(n - 1)
  END IF
END FUNCTION
```

Fibonacci Using Recursion:

```
FUNCTION fibonacci(n)
  IF n == 0 THEN
    RETURN 0
  ELSE IF n == 1 THEN
    RETURN 1
  ELSE
    RETURN fibonacci(n - 1) + fibonacci(n - 2)
  END IF
END FUNCTION
```

Common Algorithms in Pseudo Code

1. **Finding the Maximum Number in a List**:

```
FUNCTION findMax(list, n)
  max = list[0]
```

```
    FOR i = 1 TO n DO
        IF list[i] > max THEN
            max = list[i]
        END IF
    END FOR
    RETURN max
END FUNCTION
```

2. **Linear Search**:

```
FUNCTION linearSearch(list, key, n)
    FOR i = 0 TO n DO
        IF list[i] == key THEN
            RETURN i
        END IF
    END FOR
    RETURN -1
END FUNCTION
```

3. **Binary Search** (Array must be sorted):

```
FUNCTION binarySearch(list, key, low, high)
    WHILE low <= high DO
        mid = (low + high) / 2
        IF list[mid] == key THEN
            RETURN mid
        ELSE IF list[mid] < key THEN
            low = mid + 1
        ELSE
            high = mid - 1
        END IF
    END WHILE
    RETURN -1
END FUNCTION
```

Key Logical Operators

- **AND**: Both conditions must be true.

```
IF x > 0 AND y > 0 THEN
```

- **OR**: At least one condition must be true.

```
IF x > 0 OR y > 0 THEN
```

- **NOT**: Reverses the logical value.

```
IF NOT x > 0 THEN
```

Tips for Writing Pseudo Code

1. **Be Clear & Concise**: Keep the logic straightforward. Focus on clarity rather than precision.
2. **Use Proper Indentation**: Indent blocks of code for better readability.
3. **Start with the Base Case for Recursion**: Always define the stopping condition for recursion.
4. **Dry Run Your Code**: Walk through your pseudo-code with an example to ensure it behaves as expected.

Example Pseudo-Code Problems

1. **Find Factorial**:

```
FUNCTION factorial(n)
  IF n == 0 THEN
    RETURN 1
  ELSE
    RETURN n * factorial(n - 1)
  END IF
END FUNCTION
```

2. **Sum of Numbers**:

```
FUNCTION sum(n)
  total = 0
  FOR i = 1 TO n DO
    total = total + i
  END FOR
  RETURN total
END FUNCTION
```

3. **Check Even or Odd**:

```
FUNCTION isEven(n)
  IF n % 2 == 0 THEN
    RETURN True
  ELSE
    RETURN False
  END IF
END FUNCTION
```

Pseudo Code Formatting Reminders

- **Comments**: Include comments to explain complex parts of your logic.

```
// This function checks if a number is even
```

- **Whitespace**: Use blank lines between blocks of code to improve readability.

By following this cheat sheet, you'll have a solid foundation for tackling pseudo-code questions in assessments. Remember to practice recursion and common algorithms, as they are frequently tested