LANGUAGE ANALYSIS REPORT

1.**Language choice and justification**

a). Language: python

Reasons:

- Readability & Simplicity: Python's clean syntax makes the code easy to understand and maintain.
- like sqrt () and power (**), reducing manual implementation.
- Error Handling: Python’s try-except blocks make input validation and error handling straightforward.

B). What Influenced My Decision to Choose Python?

Readability and Maintainability

- Python’s clean, English-like syntax makes the code easy to write, debug, and modify.

c). Is this language suitable for this type of application?

Yes, Python is highly suitable for this calculator because:

- It efficiently handles user input/output with simple functions like input() and print().
- Its dynamic typing simplifies variable handling (no need to declare types explicitly).
- The built-in exception handling ensures robustness against invalid inputs (e.g., division by zero, non-numeric entries).
- No compilation step means faster testing and debugging.

2. Language Evaluation Criteria Analysis

a). Readability

Excellent – Python is designed for clarity.

- Minimal syntax clutter (no semicolons, braces)
- Meaningful indentation enforces structure
- Self-documenting code with intuitive function names

Example (Good Readability):

```
def calculate_percentage (a, b):
    """Calculates a% of b"""
    return (a * b) / 100  # Clear intent
```

Example (Poor Readability in Other Languages):

java

public double calculatePercentage(double a, double b) {

return (a * b) / 100.0; *// More verbose*}

Features Helping Readability:

- Dynamic typing (no type declarations cluttering logic)
- English-like keywords (def, try-except)

Hindrances:

- Weak type hints (optional, but lack of enforcement can cause subtle bugs)

b). Writability

Excellent – Rapid development with concise code.

Features making it easier to write.

- **High-level abstractions** (e.g., ** for exponentiation)
- **Rich standard library** (e.g., math module)

Python's expressiveness allowed me to implement the calculator with minimal lines of code while maintaining readability.

c). Reliability

Good but Not Perfect – Trade-offs in type safety.

Weak type checking (e.g., "5" + 3 raises Type Error at runtime, not compile-time).

Error Handling Approach:

- Used try-except for all user inputs and math operations.

d). Cost

Development Time:

- <1 hour to implement (thanks to Python's simplicity)

Development environment:

- Free tools: VS Code, PyCharm Community, IDLE
- No licensing fees (open-source)

Business Cost-Effectiveness:

- Pros: Fast development = lower labor costs
- Cons: Slower execution speed vs. C++/Java

3. Implementation Challenges and Solutions

Problem: Handling User Input Robustly

- Ensuring valid numeric inputs while allowing both integers (5) and floats (3.14).
- Preventing crashes from invalid operators or division by zero.

Helpful Features:
Dynamic Typing

- Allowed flexible input handling without type declarations.
- Example: float(input()) accepts any number format.

Exception Handling

- Clean error recovery with try-except:

Hindrances:
No Compile-Time Type Checks

- Runtime errors (e.g., "5" + 3) required explicit validation

Global State for History

- Used a global calculation_history list (risky in larger apps).
- In Java/C++, encapsulation (e.g., a history class) would be cleaner.

.If Using Another Language... Java/C++ Approach:

1. Strong Typing:
    - Compile-time checks for numbers (e.g., double x = scanner.nextDouble()).
    - No need for manual isinstance() checks.
2. Encapsulation:
    - History tracking would use a class
3. Performance:
    - Faster execution (critical for complex calculators).

Trade-Offs:

- More Boilerplate: Type declarations, Scanner setup.
- Less Flexible: Harder to handle mixed input types.

JavaScript Approach:

- Pros: Event-driven I/O (good for web calculators).
- Cons: Weak typing + no native console input.

4. **Language Comparison**

Feature	Python	Java
Typing	Dynamic (flexible but risky)	Static (strict, compile-time checks)
Syntax	Minimal boilerplate (indentation-based)	Verbose (requires classes, type declarations)
Error Handling	Simple `try-except` blocks	Complex `try-catch` with checked exceptions
Speed	Slower (interpreted)	Faster (JIT-compiled)
Input/Output	Built-in `input()`/`print()`	Requires `Scanner` and `System.out`

Python is better. Why?

- Faster development: Fewer lines of code, no compilation step.
- Built-in math support: ** for power, math.sqrt().
- Easier I/**O**: Simple input() vs. Java's Scanner boilerplate.

Specific Implementation Differences

Example 1: Input Handling

Python (3 lines):

```python
num = float(input("Enter a number: "))  # Dynamic typing
```

Java (6+ lines):

```java
import java.util.Scanner;  // Boilerplate
Scanner sc = new Scanner(System.in);
double num = sc.nextDouble();  // Static typing
```

Example 2: Power Operation

Python (1 line):

```python
result = a ** b  # Built-in operator
```

Java (Method call):

```java
double result = Math.pow(a, b);  // Verbose
```