NODE LANGUAGE - COMPLETE OVERVIEW

1. CORE PHILOSOPHY
- Strategy-oriented symbolic language
- AOT Compilation to native .exe (Windows x64)
- Proof-State Execution: deterministic and traceable
- Subversive Syntax: symbolic and expressive
- Parabase VUI-integrated

2. CHARACTER SET
Symbols: a-z, A-Z, 0-9, =, ==, +, -, *, /, %, ^, <, >, ->, <-, :, ;, |...|, ~>, <~, <<, >>
Delimiters: (), {}, [], |, #, **, "

3. PROGRAM STRUCTURE
```
Start | main |
Init message == "Hello, World!";
print(message);
Return;
```

4. SYNTAX CATEGORIES
- Declarations: Init X = 5;
- Conditionals: if X < 5 { ... } else { ... }
- Loops: for (...) { ... }, while (...) { ... }
- Operators: +, -, *, /, %, ^, ==, =, <, >, <=, >=, and, or, xor, not
- Macros: | macro_name | ... Return;

5. SEMANTICS
- throw: error jump
- proof/truth/state: validation system
- checkpoint/rollback: save/load execution
- async/await: scheduler
- this.: self-scope
- : BaseClass: inheritance

6. DATA TYPES
Int, Val, Var, Param, Init

7. ADVANCED STRUCTURES
```
class Logger {
public message == "Starting...";
log() {
print(this.message);
}
}
```

8. COMPILATION MODEL
1. Lexing
2. Symbol Resolution
3. Macro Expansion
4. Proof Validation
5. NASM Generation
6. Assembly & Linking

9. OUTPUT & EXECUTION
- Pure .exe files
- No runtime VM
- Sim + Compile hybrid execution

10. EXAMPLE: LOGIC WITH AGI
```
Start | validate |
xor_eq(A, B) : result;
if not_eq(result, 0) {
throw;
}
Return;
```

Expands to NASM:
```
XOR RAX, RBX
CMP RAX, 0
JNE throw_handler
```

11. COMPILER TOOLCHAIN
- NODECompiler.exe
- NODE.tmLanguage.json
- NODE_Language_Grammar.txt / .ebnf
- NODE_Language_Semantics_v1.0.pdf

12. TRUST & PROOF SYSTEM
truth, proof, state, verify, accept, deny, ruleset, theorem

13. PURPOSE
- Tactical systems
- Simulation engines
- Symbolic education
- Proof-audited logic
- Embedded design