Manaknight by Example

Manaknight is a **deterministic, capability-based, functional language** designed for auditable systems, APIs, and sandboxed execution.

If it compiles, it behaves **exactly** as written.

---

1. Hello World (Pure Function)

```
function hello(): String {
  "Hello, Manaknight"
}
```

Rules:

- Last expression is returned
- No `return` keyword
- No side effects

---

2. Variables & Immutability

```
let x = 10
let y = x + 5
```

❌ This is illegal:

```
x = 20   // compile error
```

Rules:

- All bindings are immutable
- Reassignment is forbidden
- State changes = new values

---

3. Functions

```
function add(a: Int, b: Int): Int {
  a + b
}
```

Rules:

- Parameters are typed
- Return type is explicit or inferred
- Functions are pure by default

---

4. If Expressions (Total by Construction)

```
function classify(age: Int): String {
  if age >= 18 {
    "adult"
  } else {
    "minor"
  }
}
```

Rules:

- `if` **must** have `else`
- Both branches must return the same type
- No implicit `void` or `Unit`

---

5. Pattern Matching

```
type Option<T> {
  | some(value: T)
  | none
}

function unwrap(o: Option<Int>): Int {
  match o {
```

```
      some(v) -> v
      none    -> 0
    }
  }
```

Rules:

- Matches must be exhaustive
- Missing cases = compile error
- No default fallthrough

---

6. Algebraic Data Types (ADTs)

```
type Payment {
  | Card(number: String)
  | Wire(reference: String)
}
```

Usage:

```
function describe(p: Payment): String {
  match p {
    Card(n) -> "Card " + n
    Wire(r) -> "Wire " + r
  }
}
```

Why:

- No inheritance
- No runtime type checks
- Compiler enforces completeness

---

7. Lists (Immutable & Recursive)

```
type List<T> {
  | cons(head: T, tail: List<T>)
```

```
  | nil
}
```

Example:

```
let numbers = cons(1, cons(2, cons(3, nil)))
```

Standard library provides:

- `map`
- `filter`
- `fold`
- `length`

---

8. Maps (Deterministic)

```
let users = set(emptyMap(), 1, "Alice")
let users2 = set(users, 2, "Bob")
```

Rules:

- Maps are immutable
- Equality is structural
- Iteration order is deterministic but not insertion-based

---

9. Effects (Capabilities, Not Magic)

Declare effects explicitly:

```
effect http
```

Use them explicitly:

```
function fetchPing(): String uses { http } {
  http.get("/ping")
}
```

Rules:

- Undeclared effects = compile error
- Pure functions cannot call effectful ones
- Lambdas are always pure

---

10. APIs (Language Feature)

```
api GET /users/:id {
  uses { db }

  let user = db.findUser(id)
  respond json(user)
}
```

What the runtime provides automatically:

- Routing
- Parameter parsing
- Type validation
- Effect scoping
- OpenAPI generation

No frameworks required.

---

11. Error Handling with `Result`

```
type Result<T, E> {
  | ok(value: T)
  | err(error: E)
}
```

Example:

```
function divide(a: Int, b: Int): Result<Int, String> {
  if b == 0 {
    err("divide by zero")
  } else {
    ok(a / b)
```

```
  }
}
```

Handling:

```
match divide(10, 2) {
  ok(v)  -> v
  err(e) -> 0
}
```

Rules:

- No exceptions
- No panics
- Errors are explicit data

---

12. Pipelines

```
value |> f |> g |> h
```

Equivalent to:

```
h(g(f(value)))
```

Rules:

- Evaluated left-to-right
- No hidden side effects

---

13. Modules

```
module auth.user {
  export login

  function login(name: String): Bool {
    true
  }
}
```

Rules:

- No executable top-level code
- Explicit exports only
- Imports are static

---

14. What Is Explicitly Forbidden

Manaknight does **not** support:

- Mutation
- `null` / `undefined`
- Exceptions
- Classes & inheritance
- Reflection or introspection
- Macros or metaprogramming
- Dynamic code loading
- Shared mutable state
- Implicit coercions
- Concurrency primitives

These are deliberate design choices.

---

15. Mental Model Summary

If you are used to:

- **Node.js** → Manaknight removes runtime ambiguity
- **Go** → Manaknight removes hidden state & IO
- **Rust** → Manaknight removes memory & lifetime complexity
- **WASM** → Manaknight adds high-level semantics

> Manaknight trades flexibility for **determinism, auditability, and safety**.

Final Note

Manaknight is designed for systems where:

- correctness matters
- behavior must be explainable
- execution must be reproducible

If it compiles, it is:
✓ deterministic
✓ auditable
✓ sandbox-safe