Exercise 1.1: What is a Pulse?

⏱️ **Time**: 15 minutes
📚 **Level**: 1 - Pulse Basics
🎯 **Prerequisite**: None (start here!)

🎯 What You'll Learn

- What a Pulse is in the Intention Space model
- The three parts of a Pulse: Prompt, Response, Trivalence
- How to structure pulse responses (simple vs structured)
- Using helper functions to work with pulses
- Why Pulses are the foundation of everything

🌍 Real-World Example: A Restaurant Menu Item

Imagine you're ordering food at a restaurant. You ask:

Question: "What dish would you like?"

The menu item can give you different **information**:

- Dish name: "Vegetable Biryani"
- Price: "$12.99"
- Chef: "Kumar"
- Cook time: "30 minutes"

In Intention Space, this question-and-answer bundle is called a **Pulse**.

📖 Understanding a Pulse

A **Pulse** is the smallest unit of data in Intention Space. It has three parts:

1. **Prompt** (The Question/Identifier)

A unique name that identifies what this data represents.

Example prompts:

- `"light_switch_state"`
- `"user_name"`
- `"dish_selected"`

2. **Response** (The Answer/Data)

The actual data value(s). Can be:

- A simple single value: `["on"]`
- Structured data with multiple fields (using META rows)

3. **Trivalence** (The State)

Tells us HOW this data can be used. Three possible values:

Value	Meaning	Color	Usage
`"Y"`	**Yes** - Editable	Green	User can change this
`"N"`	**No** - Read-only	Red	Display only, cannot change
`"UN"`	**Unknown** - Action	Orange	Button/action to trigger

Pulse Types

Type 1: Simple Pulse (Single Value)

When you only need one piece of data:

```
// A light switch pulse
const lightSwitch = {
  prompt: "light_switch_state",
  responses: ["on"],           // Simple single value
  trivalence: "Y"              // User can toggle it
};
```

Structure: `responses: ["value"]` - just an array with one string.

Type 2: Structured Pulse (Multiple Fields)

When you need multiple related pieces of data:

```
// A restaurant dish pulse
const dishSelected = {
  prompt: "dish_selected",
  responses: [
    ["META", "name", "price", "chef", "cook_time"],  // Field names
    ["Vegetable Biryani", "$12.99", "Kumar", "30 min"]  // Field values
  ],
  trivalence: "Y"
};
```

Structure:

- First row: `["META", field1, field2, ...]` - defines the structure
- Second row: `[value1, value2, ...]` - the actual data

- All values are **strings**

Think of it like a mini spreadsheet:

```
| name              | price   | chef  | cook_time |
|-------------------|---------|-------|-----------|
| Vegetable Biryani | $12.99  | Kumar | 30 min    |
```

🔧 Helper Functions

To work with pulses easily, we use helper functions:

```
// ============================================================
// PULSE HELPER FUNCTIONS
// ============================================================

/**
 * Check if a pulse is simple (single value)
 */
function isSimplePulse(pulse) {
  return pulse.responses.length === 1 &&
         !Array.isArray(pulse.responses[0]);
}

/**
 * Check if a pulse is structured (has META rows)
 */
function isStructuredPulse(pulse) {
  return pulse.responses.length > 0 &&
         Array.isArray(pulse.responses[0]) &&
         pulse.responses[0][0] === "META";
}

/**
 * Get value from simple pulse
 */
function getSimpleValue(pulse) {
  if (!isSimplePulse(pulse)) {
    throw new Error("Not a simple pulse");
  }
  return pulse.responses[0];
}

/**
 * Get a specific field value from structured pulse
 */
function getFieldValue(pulse, fieldName) {
  if (!isStructuredPulse(pulse)) {
    throw new Error("Not a structured pulse");
```

```
  }

  // responses[0] is ["META", field1, field2, ...]
  // responses[1] is [value1, value2, ...]

  const meta = pulse.responses[0];
  const data = pulse.responses[1];

  // Find which position this field is in
  const fieldIndex = meta.indexOf(fieldName);

  if (fieldIndex === -1 || fieldIndex === 0) {
    return null;  // Field not found (index 0 is "META")
  }

  return data[fieldIndex - 1];  // -1 because meta has "META" at index 0
}

/**
 * Convert structured pulse to a JavaScript object
 */
function toObject(pulse) {
  if (!isStructuredPulse(pulse)) {
    throw new Error("Not a structured pulse");
  }

  const meta = pulse.responses[0].slice(1);  // Remove "META"
  const data = pulse.responses[1];

  const obj = {};
  meta.forEach((field, idx) => {
    obj[field] = data[idx];
  });

  return obj;
}
```

Examples with Helper Functions

Example 1: Simple Pulse

```
const lightSwitch = {
  prompt: "light_switch_state",
  responses: ["on"],
  trivalence: "Y"
};

// Check type
console.log(isSimplePulse(lightSwitch));  // true
console.log(isStructuredPulse(lightSwitch));  // false
```

```
// Get value
const state = getSimpleValue(lightSwitch);
console.log(state);  // "on"
```

Example 2: Structured Pulse

```
const dish = {
  prompt: "dish_selected",
  responses: [
    ["META", "name", "price", "chef", "cook_time"],
    ["Vegetable Biryani", "$12.99", "Kumar", "30 min"]
  ],
  trivalence: "Y"
};

// Check type
console.log(isSimplePulse(dish));  // false
console.log(isStructuredPulse(dish));  // true

// Get specific fields
const dishName = getFieldValue(dish, "name");
console.log(dishName);  // "Vegetable Biryani"

const price = getFieldValue(dish, "price");
console.log(price);  // "$12.99"

const chef = getFieldValue(dish, "chef");
console.log(chef);  // "Kumar"

// Convert to object
const dishObj = toObject(dish);
console.log(dishObj);
// {
//   name: "Vegetable Biryani",
//   price: "$12.99",
//   chef: "Kumar",
//   cook_time: "30 min"
// }
```

🔧 Your Turn: Create Your First Pulses

Step 1: Create Helper Functions File

Create a file called `pulse-helpers.js` and copy the helper functions above.

Step 2: Create Your Pulses

Create a file called `my-first-pulses.js`:

```
// my-first-pulses.js

// 🔧 TODO: Copy helper functions here or import them
// (For now, just copy-paste them at the top of this file)

// ============================================================
// YOUR PULSES
// ============================================================

// 🔧 TASK 1: Create a simple pulse for a door lock
const doorLock = {
  prompt: "",           // 🔧 TODO: Give it a name like "door_lock_state"
  responses: [],        // 🔧 TODO: Is it "locked" or "unlocked"?
  trivalence: ""        // 🔧 TODO: Can we change it? (Y/N/UN)
};

// 🔧 TASK 2: Create a structured pulse for a book
const bookInfo = {
  prompt: "",           // 🔧 TODO: Name it "book_info"
  responses: [
    [],                 // 🔧 TODO: META row with fields: title, author, year,
price
    []                  // 🔧 TODO: Data row with your favorite book info
  ],
  trivalence: ""        // 🔧 TODO: Can we edit book info?
};

// ============================================================
// TEST YOUR PULSES
// ============================================================

console.log("=== DOOR LOCK ===");
console.log("Is simple?", isSimplePulse(doorLock));
console.log("Value:", getSimpleValue(doorLock));

console.log("\n=== BOOK INFO ===");
console.log("Is structured?", isStructuredPulse(bookInfo));
console.log("Title:", getFieldValue(bookInfo, "title"));
console.log("Author:", getFieldValue(bookInfo, "author"));
console.log("As object:", toObject(bookInfo));
```

Step 3: Complete the Solution

▶ Click to see the solution

```
// my-first-pulses.js
// ☑ COMPLETE SOLUTION

// [Helper functions here - copy from above]
```

```
// ============================================================
// YOUR PULSES
// ============================================================

// Task 1: Simple pulse for door lock
const doorLock = {
  prompt: "door_lock_state",
  responses: ["locked"],
  trivalence: "Y"  // User can lock/unlock
};

// Task 2: Structured pulse for a book
const bookInfo = {
  prompt: "book_info",
  responses: [
    ["META", "title", "author", "year", "price"],
    ["1984", "George Orwell", "1949", "$15.99"]
  ],
  trivalence: "N"  // Book info is read-only
};

// ============================================================
// TEST YOUR PULSES
// ============================================================

console.log("=== DOOR LOCK ===");
console.log("Is simple?", isSimplePulse(doorLock));  // true
console.log("Value:", getSimpleValue(doorLock));  // "locked"

console.log("\n=== BOOK INFO ===");
console.log("Is structured?", isStructuredPulse(bookInfo));  // true
console.log("Title:", getFieldValue(bookInfo, "title"));  // "1984"
console.log("Author:", getFieldValue(bookInfo, "author"));  // "George Orwell"
console.log("As object:", toObject(bookInfo));
// {
//   title: "1984",
//   author: "George Orwell",
//   year: "1949",
//   price: "$15.99"
// }
```

☑ Check Your Understanding

Question 1: What are the three parts of a Pulse?

▶ Click to reveal answer

1. **Prompt** - the question/identifier
2. **Response** - the answer(s)/data
3. **Trivalence** - the state (Y/N/UN)

Question 2: When should you use a structured pulse instead of a simple pulse?

▶ Click to reveal answer

Use a **structured pulse** when you need multiple related pieces of data that belong together (like dish name + price + chef). Use a **simple pulse** when you only need a single value (like "on" or "off").

Question 3: What does the META row do?

▶ Click to reveal answer

The META row defines the **field names** (column headers) for the data. It tells you what each value in the data row represents.

Example:

```
["META", "name", "price"]  // Defines: first value is name, second is price
["Biryani", "$12.99"]      // Data: name="Biryani", price="$12.99"
```

Practice Challenge

Challenge 1: Traffic Light

Create a simple pulse for a traffic light:

```
// 🔧 YOUR TASK: Create a pulse for a traffic light
// Hint: Current color is "red"
// Hint: Traffic lights change automatically (not by user)

const trafficLight = {
  prompt: "",       // 🔧 TODO
  responses: [],    // 🔧 TODO
  trivalence: ""    // 🔧 TODO
};

// Test it
console.log("Traffic light color:", getSimpleValue(trafficLight));
```

▶ Solution

```
const trafficLight = {
  prompt: "traffic_light_color",
  responses: ["red"],
  trivalence: "N"   // User cannot change traffic light!
};
```

```
console.log("Traffic light color:", getSimpleValue(trafficLight));
// "red"
```

Challenge 2: Shopping Cart Item

Create a structured pulse for a shopping cart item:

```
// 🔧 YOUR TASK: Create a pulse for a cart item
// Hint: Fields needed: product_id, name, quantity, price
// Hint: Example item: ID="p001", Name="Laptop", Qty="1", Price="$999"
// Hint: User can edit quantities

const cartItem = {
  prompt: "",
  responses: [
    [],  // 🔧 TODO: META row
    []   // 🔧 TODO: Data row
  ],
  trivalence: ""
};

// Test it
console.log("Product:", getFieldValue(cartItem, "name"));
console.log("Quantity:", getFieldValue(cartItem, "quantity"));
console.log("As object:", toObject(cartItem));
```

▶ Solution

```
const cartItem = {
  prompt: "cart_item",
  responses: [
    ["META", "product_id", "name", "quantity", "price"],
    ["p001", "Laptop", "1", "$999"]
  ],
  trivalence: "Y"  // User can edit quantities
};

console.log("Product:", getFieldValue(cartItem, "name"));
// "Laptop"

console.log("Quantity:", getFieldValue(cartItem, "quantity"));
// "1"

console.log("As object:", toObject(cartItem));
// {
//   product_id: "p001",
//   name: "Laptop",
//   quantity: "1",
```

```
//   price: "$999"
// }
```

Challenge 3: User Profile

Create a structured pulse for a user profile:

```
// 🔧 YOUR TASK: Create a pulse for user profile
// Hint: Fields: user_id, name, email, role
// Hint: Make up your own data!
// Hint: User can edit their profile

const userProfile = {
  prompt: "",
  responses: [
    [],  // 🔧 TODO
    []   // 🔧 TODO
  ],
  trivalence: ""
};

// Test it with helper functions
console.log("User:", getFieldValue(userProfile, "name"));
console.log("Email:", getFieldValue(userProfile, "email"));
```

▶ Solution

```
const userProfile = {
  prompt: "user_profile",
  responses: [
    ["META", "user_id", "name", "email", "role"],
    ["u123", "Alice Smith", "alice@example.com", "admin"]
  ],
  trivalence: "Y"  // User can edit their profile
};

console.log("User:", getFieldValue(userProfile, "name"));
// "Alice Smith"

console.log("Email:", getFieldValue(userProfile, "email"));
// "alice@example.com"
```

Key Takeaways

After completing this exercise, you should understand:

☑ **A Pulse is a question-answer pair** with a state indicator

☑ **Three parts**:

- Prompt = unique identifier
- Response = data (simple or structured)
- Trivalence = usage indicator (Y/N/UN)

☑ **Two types of pulses**:

- Simple: `responses: ["value"]`
- Structured: `responses: [["META", ...], [data, ...]]`

☑ **Helper functions** make working with pulses easy:

- `isSimplePulse()` / `isStructuredPulse()`
- `getSimpleValue()` / `getFieldValue()`
- `toObject()` for structured pulses

☑ **Pulses are self-describing** - the META row tells you what each value means

☑ **All values are strings** - the platform can parse them later if needed

---

🎨 Visual Summary

```
SIMPLE PULSE
┌─────────────────────────────────────┐
│  Prompt: "light_switch_state"       │
│  Response: ["on"]                   │
│  Trivalence: "Y"                    │
└─────────────────────────────────────┘

STRUCTURED PULSE (Think: Mini Spreadsheet)
┌─────────────────────────────────────────────────────┐
│  Prompt: "dish_selected"                            │
│  Response:                                          │
│    ["META", "name", "price", "chef"]  ← Column headers │
│    ["Biryani", "$12.99", "Kumar"]     ← Data row    │
│  Trivalence: "Y"                                    │
└─────────────────────────────────────────────────────┘
```

🧪 Run Your Code

To test your pulses:

```
# Save your code as my-first-pulses.js
# Run it with Node.js
node my-first-pulses.js
```

Quick Reference Card

Keep this handy for future exercises:

```
// Simple Pulse Template
const simplePulse = {
  prompt: "unique_identifier",
  responses: ["value"],
  trivalence: "Y|N|UN"
};

// Structured Pulse Template
const structuredPulse = {
  prompt: "unique_identifier",
  responses: [
    ["META", "field1", "field2", "field3"],
    ["value1", "value2", "value3"]
  ],
  trivalence: "Y|N|UN"
};

// Helper Functions Quick Guide
isSimplePulse(pulse)          // true/false
isStructuredPulse(pulse)      // true/false
getSimpleValue(pulse)         // "value"
getFieldValue(pulse, "field") // "value"
toObject(pulse)               // { field1: "value1", ... }

// Trivalence Quick Guide
"Y"  = Editable (🟢 Green)
"N"  = Read-only (🔴 Red)
"UN" = Action (🟠 Orange)
```

Next Exercise

Now that you understand what a Pulse is and how to work with them, let's see how Pulses **change over time**.

Next: Exercise 1.2: Pulses Change Over Time

In the next exercise, you'll learn:

- How to update pulse responses
- When and why pulses change
- The lifecycle of a pulse
- Creating functions to modify pulses

Bonus: Behind the Scenes

▶ 🤓 Why use META rows instead of JSON objects?

Question: Why not just use JavaScript objects like this?

```
responses: {
  name: "Biryani",
  price: "$12.99"
}
```

Answer: The Response Array Convention uses arrays because:

1. **Platform Independent**: Arrays work the same in all languages (JavaScript, Python, Go, etc.)
2. **Multiple Records**: Arrays easily support multiple rows of data
3. **Ordered**: Field order is preserved (objects don't guarantee order in all languages)
4. **Simple Parsing**: Easy to parse row-by-row (like CSV files)
5. **Future-Proof**: Can add more meta rows for different data structures

The convention is designed to work across ALL platforms, not just JavaScript!

▶ 🤓 Why are all values strings?

Question: Why store numbers as strings like `"25"` instead of `25`?

Answer:

1. **Universal**: Strings work everywhere (databases, files, network, any language)
2. **Safe**: No type conversion errors during data transfer
3. **Flexible**: Platform decides how to interpret (`"25"` → number, `"true"` → boolean)
4. **Convention**: The Response Array Convention defines the structure, not the types

When you use the data, you can parse it:

```
const age = parseInt(getFieldValue(pulse, "age"));  // "25" → 25
const price = parseFloat(getFieldValue(pulse, "price"));  // "$12.99" → ?
```

This separation keeps the convention simple and universal!

🎉 **Congratulations!** You've completed Exercise 1.1. You now understand:

- What Pulses are
- How to structure pulse responses
- How to use helper functions to work with pulses

You're ready for Exercise 1.2!

Last Updated: December 22, 2024
Exercise Series: Intention Tunnel for Beginners
License: Educational use