Lab Exercise 2 - Designing Prompts for Structured Data

---

1. Objective

This lab focuses on **prompt design techniques** to make ChatGPT return **structured outputs**, specifically:

- Valid **JSON responses**
- **Schema-controlled outputs**
- **Function-call–style responses** suitable for API integration

By the end of this lab, learners will be able to design prompts that reliably produce machine-readable data instead of free-form text.

---

2. Prerequisites

- Basic understanding of JSON
- Familiarity with APIs or backend systems (helpful but not mandatory)
- Access to ChatGPT (web or API)

---

3. Why Structured Prompts Are Important

Structured prompting is required when:

- Integrating ChatGPT with applications
- Automating workflows
- Passing outputs to other systems
- Avoiding ambiguity in responses

Examples:

- Returning JSON for APIs
- Extracting entities from text
- Triggering backend functions

---

4. Part A: Prompting for JSON Output

Task

Generate customer data strictly in JSON format.

Prompt

```
You are an API.

Return ONLY valid JSON.

Do not add explanations.

Generate customer details with fields:

- customerId (string)

- name (string)

- email (string)

- totalPurchase (number)
```

Expected Output

```
{

"customerId": "C001",

"name": "John Doe",

"email": "john.doe@example.com",

"totalPurchase": 1250.75
}
```

Learning

- Explicitly instructing *"Return ONLY JSON"* avoids extra text
- Field names and data types must be specified

---

5. Part B: Enforcing a JSON Schema

Task

Ensure ChatGPT follows a fixed schema.

Prompt

```
Return output in the following JSON schema only:

{

"orderId": string,

"customerId": string,

"items": [

{

"product": string,

"quantity": number,

"price": number

}

],

"totalAmount": number

}


Generate sample order data.
```

Expected Output

```
{
"orderId": "O1001",
"customerId": "C001",
"items": [
{
"product": "Laptop",
"quantity": 1,
"price": 1200
}
],
"totalAmount": 1200
}
```

Learning

- Providing a schema strongly guides output structure
- Reduces parsing errors in applications

---

6. Part C: Prompting for Arrays of JSON Objects

Task

Generate multiple records in JSON array format.

Prompt

```
Return ONLY a JSON array.
Each object must contain:
```

```
- id

- name

- country


Generate 3 customer records.
```

Expected Output

```
[

{“id”: “C001”, “name”: “Amit”, “country”: “India”},

{“id”: “C002”, “name”: “Sarah”, “country”: “USA”},

{“id”: “C003”, “name”: “Luis”, “country”: “Spain”}

]
```

---

7. Part D: Prompting for Function-Call-Style Output

Concept

Function calling allows ChatGPT to return **arguments** needed to invoke backend logic.

Task

Extract intent and parameters for a function call.

Function Definition (Conceptual)

```
function createOrder(customerId, product, quantity)
```

Prompt

```
You are a system that prepares function calls.

Return ONLY JSON arguments for the function below.

Function: createOrder(customerId, product, quantity)

User input:

"Create an order for customer C001 for 2 laptops"
```

Expected Output

```
{

"customerId": "C001",

"product": "laptop",

"quantity": 2

}
```

Learning

- No natural language explanation
- Output maps directly to function parameters

---

8. Part E: Validating and Restricting Output

Task

Prevent invalid values.

Prompt

```
Return JSON only.

Ensure quantity is a positive integer.

If input is invalid, return:



{ "error": "Invalid input" }



User input:

"Order minus 3 phones for customer C005"
```

Expected Output

```
{ "error": "Invalid input" }
```

---

9. Part F: Combining Explanation + Structured Output (Controlled)

Task

Return JSON and a short explanation in separate fields.

Prompt

```
Return output as JSON with two fields:

- data (object)

- explanation (string)



Generate sample login event data.
```

Expected Output

```
{

"data": {

"userId": "U123",

"loginTime": "2024-06-01T10:30:00Z",

"ipAddress": "192.168.1.10"

},

"explanation": "This record represents a user login event."

}
```

---

10. Common Prompting Best Practices

- Clearly say **"Return ONLY JSON"**
- Define field names and data types
- Provide sample schema when possible
- Avoid ambiguous instructions
- Use capitalized constraints (ONLY, MUST)
- Validate output before using in production

---

11. Common Mistakes to Avoid

- Forgetting to restrict extra text
- Not specifying array vs object
- Leaving data types undefined
- Mixing explanation with data unintentionally

---

12. Conclusion

Designing prompts for structured data is essential for real-world ChatGPT integrations. By explicitly defining schemas, constraints, and output rules, developers can reliably use ChatGPT for JSON generation, API workflows, and function calling without manual cleanup.