JSON Schema Overview and Examples

JSON Schema is a tool to define and validate the structure of JSON data. It ensures that the data adheres to specific types, structures, and constraints, making it useful for APIs and data validation.

Basic JSON Schema Example

This schema defines a “Person” object with properties `name`, `age`, and `email`.

```
{
  "$schema": "http://json-schema.org/draft-07/schema#",
  "title": "Person",
  "type": "object",
  "properties": {
    "name": { "type": "string" },
    "age": { "type": "integer", "minimum": 0 },
    "email": { "type": "string", "format": "email" }
  },
  "required": ["name", "email"]
}
```

Explanation:

- `name` and `email` are required.

- `age` is optional but must be a non-negative integer if provided.

- `email` should be a valid email format.

Valid Example for this Schema:

```
{
  "name": "John Doe",
  "age": 28,
  "email": "john@example.com"
}
```

Arrays in JSON Schema

You can validate arrays by specifying the type of items and enforcing constraints like `minItems` or `uniqueItems`.

Array Schema Example

```
{
  "type": "array",
  "items": { "type": "string" },
  "minItems": 1,
  "uniqueItems": true
}
```

This schema defines an array where:

- Items must be strings.

- The array must have at least one item.

- All items must be unique.

Valid JSON Array:

```
[
  "apple",
  "banana",
  "cherry"
]
```

Conditional Validation in JSON Schema

JSON Schema supports conditional logic using `if`, `then`, and `else`.

Schema for Conditional Validation

```
{
  "type": "object",
  "properties": {
    "isEmployed": { "type": "boolean" }
  },
  "if": {
    "properties": { "isEmployed": { "const": true } }
  },
  "then": {
    "properties": {
      "employer": { "type": "string" }
    },
    "required": ["employer"]
  },
  "else": {
    "properties": {
      "reasonUnemployed": { "type": "string" }
```

```
    },
    "required": ["reasonUnemployed"]
  }
}
```

Explanation:

- If `isEmployed` is `true`, `employer` must be provided.

- If `isEmployed` is `false`, `reasonUnemployed` must be provided.

Valid Example (Employed):

```
{
  "isEmployed": true,
  "employer": "TechCorp"
}
```

Valid Example (Unemployed):

```
{
  "isEmployed": false,
  "reasonUnemployed": "Student"
}
```

Summary

JSON Schema is versatile for defining the structure and validation rules for JSON data. It supports basic types, complex structures like arrays, and conditional logic, allowing developers to ensure data consistency and correctness. By using JSON Schema, you can make sure your data adheres to predefined formats, helping with integration and error prevention in APIs and applications.