JSON SCHEMAS

JSON: Javascript Object Notation

- Object:
  `{ "key1": "value1", "key2": "value2" }`
- Array:
  `[ "first", "second", "third" ]`
- Number: `42, 3.1415926`
- String: `"This is a string"`
- Boolean: `true, false`
- Null: `null`

JSON vs Java Data Types

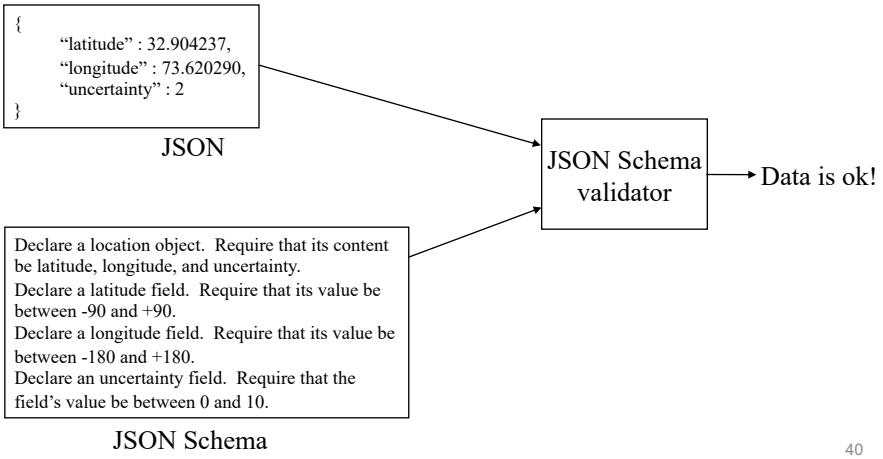
JSON	Java
string	String
integer	int
number	float
boolean	bool
object	Dictionary (Hashtable)
array	List (ArrayList)
null	Object

Note: object ≠ array in JSON!

39

39

Validating Your Data



40

40

Examples

- Example A:

```
{
  "name": "George Washington",
  "birthday": "February 22, 1732",
  "address": "Mount Vernon, Virginia, United States"
}
```

- Example B:

```
{
  "first_name": "George",
  "last_name": "Washington",
  "birthday": "1732-02-22",
  "address": {
  "street_address": "3200 Mount Vernon Memorial Highway",
  "city": "Mount Vernon",
  "state": "Virginia",
  "country": "United States"
}
```

41

41

Example Schema

```
{
  "type": "object",
  "properties": {
    "first_name": { "type": "string" },
    "last_name": { "type": "string" },
    "birthday": { "type": "string", "format": "date" },
    "address": {
      "type": "object",
      "properties": {
        "street_address": { "type": "string" },
        "city": { "type": "string" },
        "state": { "type": "string" },
        "country": { "type" : "string" }
      }
    }
  }
}
```

42

42

SIMPLE JSON SCHEMAS

43

43

Simple Schemas

- Schema: Constraints on the allowable data
- Accept anything:

```
{ }
true
```

- Accept nothing:

```
false
```

44

44

Simple Schemas

- Schema: Constraints on the allowable data
- Accept string:
```
{ "type" : "string" }
```
- Accept number (integer or float):
```
{ "type" : "number" }
```
- Accept string or number:
```
{ "type" : [ "number", "string" ] }
```

45

45

String Schemas

- Length constraints on a string
```
{
  "type": "string",
  "minLength": 2,
  "maxLength": 3
}
```
- Regular expressions (always "^...$")
```
{
  "type": "string",
  "pattern": "^(\\([0-9]{3}\\))?[0-9]{3}-[0-
9]{4}$"
}
```

"(888)555-1212"

46

46

String Formats

- Semantic specification for string values
- Validation optional and left to implementation

```
{
  "type": "string",
  ”format": "date-time"
}
```

```
"2018-11-13T20:20:39+00:00"
```

```
{
  "type": "string",
  "format": "email"
}
```

```
"foo@bar.com"
```

47

47

String Schemas and Formats

- Some formats:
  - `date-time, time, date`
  - `email`
  - `hostname`
  - `ipv4, ipv6`
  - `uuid`
  - `uri, uri-reference`
  - `json-pointer` (e.g. `"/foo/bar"`)
  - `json-pointer-reference` (e.g. `"#/foo/bar"`)

48

48

Media and Non-JSON Data

- Non-JSON data encoded as strings (e.g. images)
- Specify MIME type and content encoding (e.g/ Base64)

```
{
  "type": "string",
  "contentMediaType": "text/html"
}
```

```
"<!DOCTYPE html><html><head></head></html>"
```

```
{
  "type": "string",
  "contentEncoding": "base64",
  "contentMediaType": "image/png"
}
```

```
"iVBORw0KGgoAAAANSUhEUgAAABgAAAAYCAYAAA..."
```

49

49

Enumerated Schemas

- Enumerated list of values
- Values may be of different types (no `"type"`)

```
{
  "enum": [ "Drama",
            "History",
            "Comedy",
            "Mystery",
            "SciFi",
            "Art”
          ]
}
```

50

50

Number Schemas

- Multiples:

```
{
  "type": "integer",
  "multipleOf" : 10
}
```

- Ranges (`minimum, maximum, exclusiveMinimum, exclusiveMaximum`):

```
{
  "type": "number",
  "minimum": 0,
  "exclusiveMaximum": 100
}
```

51

51

OBJECT SCHEMAS

52

52

Object Schema

- Simply require an object

```
{ "type": "object" }
```

- Schema for specific properties:

```
{
  "type": "object",
  "properties": {
    "street": { "type": "string" },
    "zip": { "type": "[0-9]{5}" },
    "state": { "enum": ["NJ", "NY"] }
  }
}
```

```
{ "street": "Castle Point"
  { "street": "Castle Point",
    "zip": "07030",
    "state": "NJ" }
```

53

53

Pattern Properties

- Specify schema for properties based on pattern

```
{
  "type": "object",
  "patternProperties": {
    "^S_": { "type": "string" },
    "^I_": { "type": "integer" }
  }
}
```

```
{ "S_25": "This is a string" }

{ "I_0": 42 }
```

54

54

Additional Properties

- Specify schema for additional properties

```
{
  "type": "object",
  "properties": {
    "street": { "type": "string" },
    "zip":  { "type": "string”,
              "pattern": "[0-9]{5}" },
    "state": { "enum": ["NJ", "NY"] }
  },
  "additionalProperties" : false
}
```

```
{ "street": "Castle Point",
  "zip": "07030",
  "state": "NJ",
  "city": "Hoboken"
}
```

55

55

Additional Properties

- Specify schema for additional properties

```
{
  "type": "object",
  "properties": {
    "street": { "type": "string" },
    "zip":  { "type": "string”,
              "pattern": "[0-9]{5}" },
    "state": { "enum": ["NJ", "NY"] }
  },
  "additionalProperties" : { "type": "string" }
}
```

```
{ "street": "Castle Point",
  "zip": "07030",
  "state": "NJ",
  "city": "Hoboken”
}
```

56

56

Additional Properties

- Specify schema for additional properties

```
{
  "type": "object",
  "properties": {
    "street": { "type": "string" },
    "zip":  { "type": "string”,
              "pattern": "[0-9]{5}" },
    "state": { "enum": ["NJ", "NY"] }
  },
  "additionalProperties" : { "type": "string" }
}
```

```
{ "street": "Castle Point",
  "zip": "07030",

  "city": "Hoboken”
}
```

57

57

Properties: Defined, Optional Required

- **Defined**: Property has a definition in the object

  `{..., "state": "NJ", ...}`

- **Optional**: Property does not have to be present
  - This is default!

  ```
  "properties":
     { "state": { "enum": ["NJ", "NY"] }, ...}
  ```

- **Required**: Property must be present

  `"required": ["state" ]`

58

58

Required Properties

- Specify properties that are required

```
{
  "type": "object",
  "properties": {
    "street": { "type": "string" },
    "zip":  { "type": "string",
              "pattern": "[0-9]{5}" },
    "state": { "enum": ["NJ", "NY"] }
  },
  "required" : ["street", "zip", "state"]
}
```

```
{ "street": "Castle Point",
  "zip": "07030",
  "state": "NJ"
}
```

59

59

Dependent Required

- Properties that are required if others present

```
{
  "type": "object",
  "properties": {
    "name": { "type": "string" },
    "credit_card": { "type": "number" },
    "billing_address": { "type": "string" }
  },
  "required": ["name"],
  "dependentRequired": {
    "credit_card": ["billing_address"]
  }
}
```

```
{
  "name": "John Doe",
  "credit_card": 5555555555555555,
  "billing_address": "555 Debtor's Lane"
}
```

60

60

ARRAY SCHEMAS

61

61

Array Schema

- Schema:
  `{ "type": "array" }`
- Allowable values:
  - `[1, 2, 3, 4, 5]`
  - `[3, "different", { "types" : "of values" }]`

62

62

How to Validate Arrays

- As lists:
  - Arbitrary length
  - All items the same schema
  - Specify with `items`
- As tuples
  - Fixed length (object without property names)
  - Each item may have own schema
  - Specify with `prefixItems`

63

63

List Validation

- Schema:

```
{
  "type": "array",
  "items": {
    "type": "number"
  }
}
```

- Example:
  - `[1, 2, 3, 4, 5]`

64

64

Tuple Validation

- Schema (addresses):

```
{
  "type": "array",
  "prefixItems": [
    { "type": "string" },
    { "type": "string", "pattern": "[0-9]{5}" },
    { "enum": ["NJ", "NY"] }
  ]
}
```

- Examples:
  - `["Castle Point", "07030", "NJ"]`
  - `["Castle Point", "07030", "NJ", "Hoboken"]`

65

65

Tuple Validation

- Schema (addresses):

```
{
  "type": "array",
  "prefixItems": [
    { "type": "string" },
    { "type": "string", "pattern": "[0-9]{5}" },
    { "enum": ["NJ", "NY"] }
  ]
 "items": false
}
```

- Examples:
  - `["Castle Point", "07030", "NJ"]`
  - `["Castle Point", "07030", "NJ", "Hoboken"]`

66

66

Tuple Validation

- Schema (addresses):

```
{
  "type": "array",
  "prefixItems": [
    { "type": "string" },
    { "type": "string", "pattern": "[0-9]{5}" },
    { "enum": ["NJ", "NY"] }
  ]
 "items": { "type": "string" }
}
```

- Examples:
  - `["Castle Point", "07030", "NJ"]`
  - `["Castle Point", "07030", "NJ", "Hoboken"]`

67

67

STRUCTURING A SCHEMA

68

68

Specifying Schema Language

- Reference the version of JSON Schema

```
{
  "$schema":
"https://json-schema.org/draft/2020-12/schema",
   ...
}
```

69

69

Schema Identification

- Base URI: Specify absolute URI in $id property

```
{ "$schema": "...",
  "$id": "https://example.com/schemas/content",
  "type": "object",
  "properties": {
    "name": { "type": "string" },
    ...
  },
  "required": ...
}
```

70

70

Schema Identification

- JSON Pointer: Reference subschema

```
{ "$schema": "...",
  "$id": "https://example.com/schemas/content",
  "type": "object",
  "properties": {
    ”foo": { "type": "..." },
     ...
  },
  "required": ...
}
```

```
https://example.com/schemas/content#/properties/foo
```

71

71

Schema Identification

- Anchor for subschema (less common)

```
{ "$schema": "...",
  "$id": "https://example.com/schemas/content",
  "type": "object",
  "properties": {
   "foo": { "$anchor": "foo",
            "type": "..." },
     ...
  },
  "required": ...
}
```

```
https://example.com/schemas/content#foo
```

72

72

Schema Reference

- Reference (sub)schema in another schema

```
{ "$schema": "...",
  "type": "object",
  "properties": {
    "bar": {
      "$ref": "https://example.com/schemas/..."
    },
    ...
  },
  "required": ...
}
```

73

73

Local Definitions

- Reference local definition in this schema

```
{ "$schema": "...",
  "type": "object",
  "properties": {
    "bar": {
      "$ref": "#$defs/foo"
    },
    ...
  },
  "$defs": {
    "foo": { "type": "string" },
  }
}
```

74

74

Recursive Schema

- Schemas can be self-referential

```
{ "$schema": "...",
  "type": "object",
  "properties": {
    "item": true,
    "link": { "$ref": "#" }
  }
}
```

75

75

Recursive Schema

- Mutual recursion is **not** allowed

```
{ "$schema": "...",
  "$defs": {
    "foo": { "$ref": "#/$defs/bar" },
    "bar": { "$ref": "#/$defs/foo" }
  }
}
```

76

76

SCHEMA COMPOSITION

77

77

Combining Schemas

- Employee information

```
{ "$id": "https://example.com/schemas/employ",
  "type": "object",
  "properties": {
    "salary": { "type": "number" }
  }
}
```

- Teaching information

```
{ "$id": "https://example.com/schemas/teaching",
  "type": "object",
  "properties": {
    "courses": { "type": "array",
                 "items": { "type": "string" }
               }
  }
}
```

78

78

Combining Schemas

- Employee who teaches

```
{ "allOf": [
      { "$ref": "https://example.com/schemas/employ" },
      { "$ref": "https://example.com/schemas/teaching" }
  ]
}
```

79

79

Combining Schemas

- Either a student or a teacher (can be both)

```
{ "anyOf": [
      { "$ref": "https://example.com/schemas/student" },
      { "$ref": "https://example.com/schemas/teaching" }
  ]
}
```

80

80

Combining Schemas

- Either a student or an alumnus (cannot be both)

```
{ "oneOf": [
      { "$ref": "https://example.com/schemas/student" },
      { "$ref": "https://example.com/schemas/teaching" }
  ]
}
```

81

81

Combining Schemas

- Does not draw a salary

```
{ "not":
      { "$ref": "https://example.com/schemas/employ" }
}
```

82

82

SUBTYPING

83

83

Subtyping with Simple Types

- Allowable Age

```
{ "$id": "https://example.com/schemas/age",
  "type": "integer",
  "minimum": 0,
  "maximum": 125
}
```

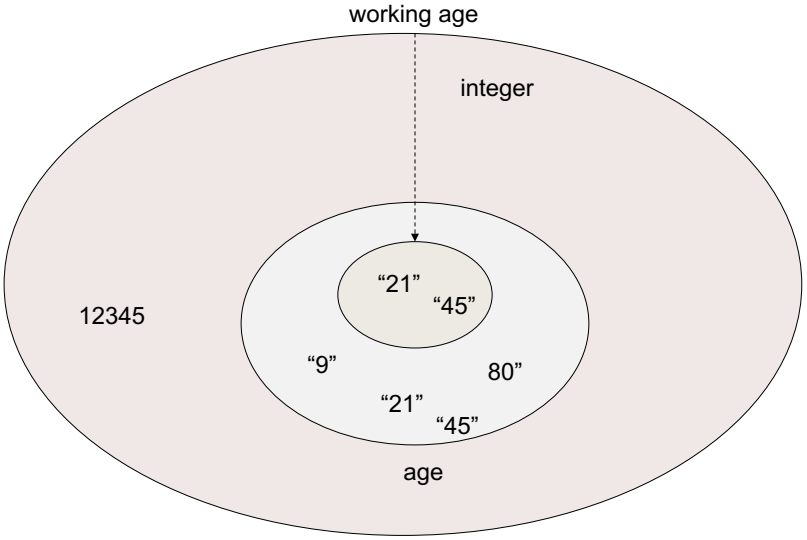
- Working Age

```
{ "allOf": [
    { "$ref": "https://example.com/schemas/age" }
  ],
  "minimum": 18,
  "maximum": 67
}
```

84

84

Simple Subtyping

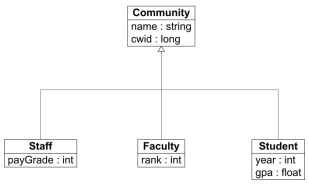


85

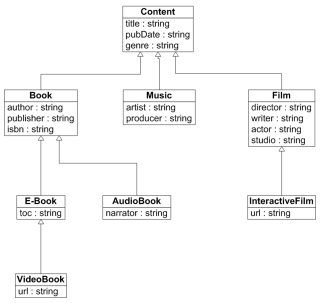
85

Data Modeling

- Generalization
  - Union of fixed set of alternatives
  - JSON Schema: oneOf / anyOf



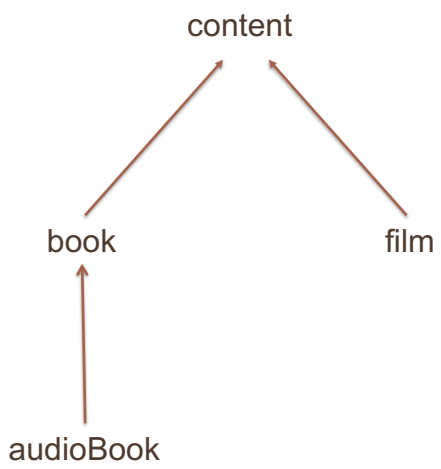
- Specialization
  - Derive open-ended set of specialized types
  - JSON Schema: allOf



86

86

Derived Object Types



87

87

Derived Object Types

- Base Schema

```
{ "$schema": "...",
  "$id": "https://example.com/schemas/content",
  "type": "object",
  "properties": {
    "title": { "type": "string" },
    "pubDate": { "type": "string", "format": "date" },
    "genre": { "enum": [ "crime", "history", ...] }
  }

}
```

```
{ "title": "Stagecoach",
  "pubDate": "1939",
  "genre": "western" }
```

88

88

Derived Object Types

- Derived Schema for Book

```
{ "$schema": "...",
  "$id": "https://example.com/schemas/book",
  "allOf": [
    { "$ref": "https://example.com/schemas/content" }
  ],
  "properties": {
    "author": { "type": "string" },
    "isbn": { "type": "integer" }
  }
}
```

```
{ "title": "Desolation Island",
  "pubDate": "2001",
  "genre": "history",
  "author": "Patrick O'Brian",
  "isbn": "9781402502248" }
```

89

89

Derived Object Types

- Derived Schema for Film

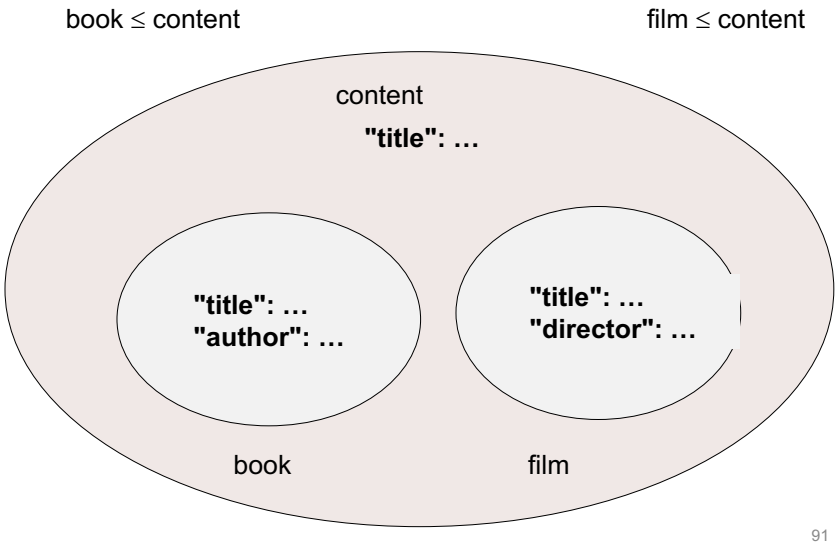
```
{ "$schema": "...",
  "$id": "https://example.com/schemas/film",
  "allOf": [
    { "$ref": "https://example.com/schemas/content" }
  ],
  "properties": {
    "writer": { "type": "string" },
    "director": { "type": "string" }
  }
}
```

```
{ "title": "Stagecoach",
  "pubDate": "1939",
  "genre": "western",
  "writer": "Dudley Nichols",
  "director": "John Ford" }
```

90

90

Complex Subtyping



91

91

Derived Object Types

- Base Schema

```
{ "$schema": "...",
  "$id": "https://example.com/schemas/content",
  "type": "object",
  "properties": {
    "title": { "type": "string" },
    "pubDate": { "type": "string", "format": "date" },
    "genre": { "enum": [ "crime", "history", ...] }
  },
  "required": [ "title", "pubDate", "genre" ]
}
```

92

92

Derived Object Types

- Derived Schema for Book

```
{ "$schema": "...",
  "$id": "https://example.com/schemas/book",
  "allOf": [
    { "$ref": "https://example.com/schemas/content" }
  ],
  "properties": {
    "author": { "type": "string" },
    "publisher": { "type": "string" },
    "isbn": { "type": "integer" }
  },
  "required": [ "author", "isbn" ]
}
```

93

93

Derived Object Types

- Derived Schema for Film

```
{ "$schema": "...",
  "$id": "https://example.com/schemas/film",
  "allOf": [
    { "$ref": "https://example.com/schemas/content" }
  ],
  "properties": {
    "writer": { "type": "string" },
    "director": { "type": "string" }
  },
  "required": [ "writer", "director" ]
}
```

94

94

Additional Properties

- Derived Schema for Film

```
{ "title": "Stagecoach" }
```

```
{ "title": "Stagecoach",
  "director": "John Ford" }
```

```
{ "allOf": [
    { "type": "object",
      "properties": {
        "title": { "type": "string" }
      },
      "required": [ "title" ],
      "additionalProperties": false
    }
  ],
  "properties": {
    "director": { "type": "string" }
  },
  "required": [ "director" ]
}
```

95

95

Additional Properties

- Derived Schema for Film

```
{ "title": "Stagecoach",
  "director": "John Ford" }
```

```
{ "allOf": [
    { "type": "object",
      "properties": {
        "title": { "type": "string" }
      },
      "required": [ "title" ],
      "unevaluatedProperties": false
    }
  ],
  "properties": {
    "director": { "type": "string" }
  },
  "required": [ "director" ]
}
```

96

96