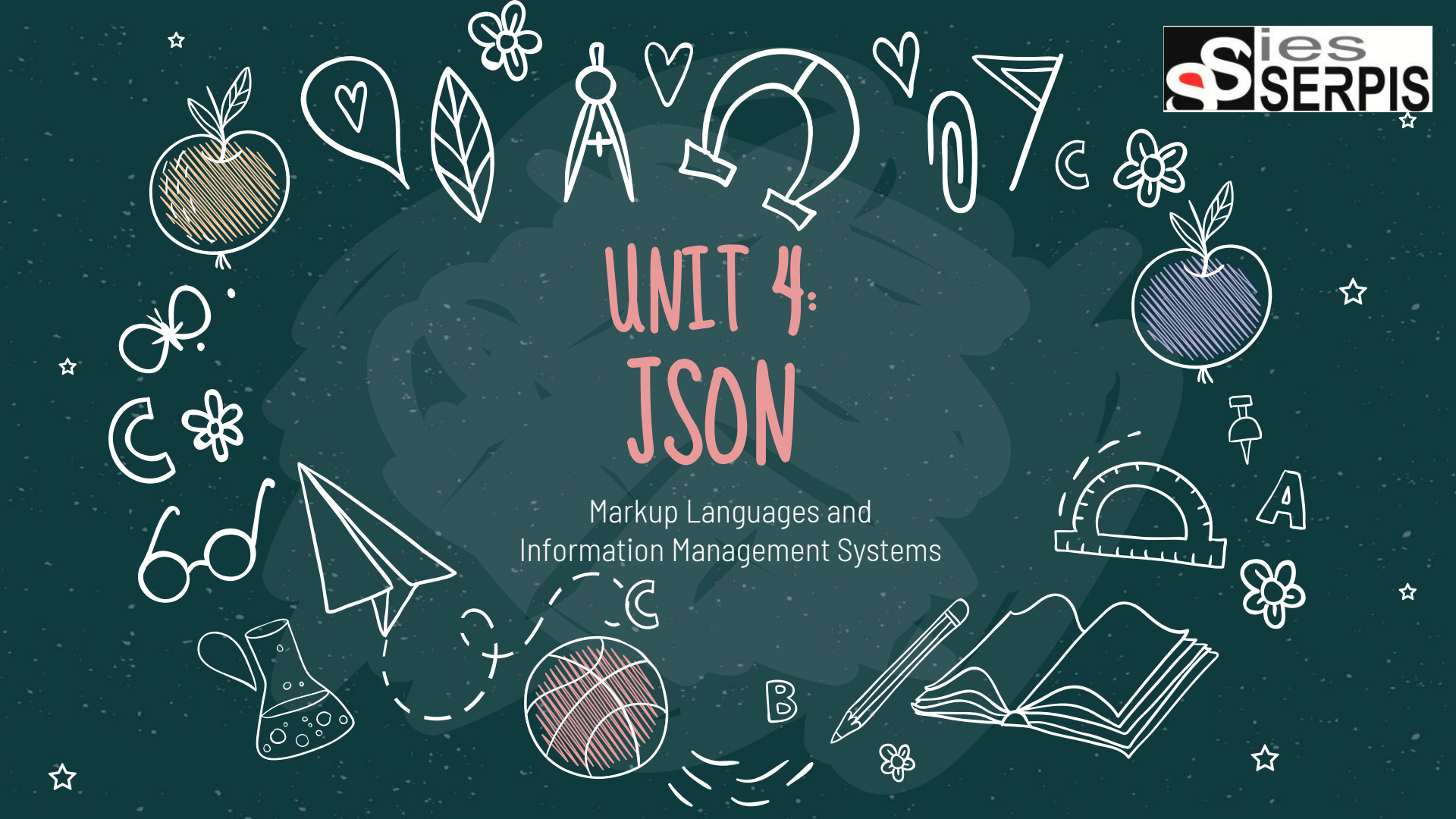

UNIT 4:
JSON
Markup Languages and
Information Management Systems
ies SERPIS

01

INTRODUCTION

02

SYNTAX

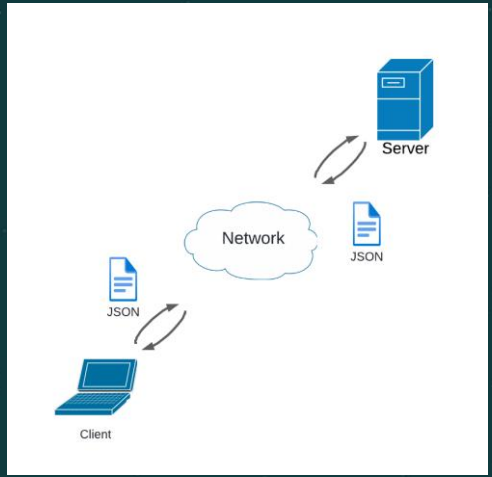
03

VALIDATION

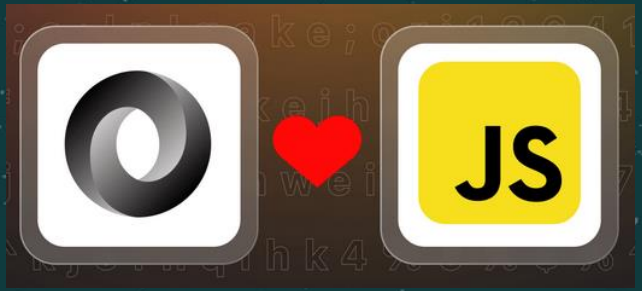
01.
INTRODUCTION

JSON
(JavaScript Object Notation)

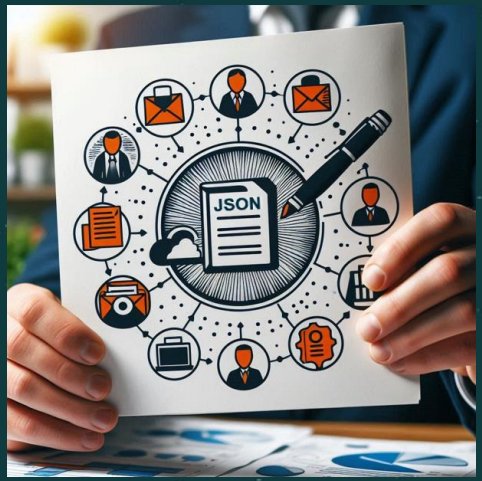
- Data-interchange format
  - **Does not use tags.**
  - Many do not consider it a *markup language*
- Born in the early 2000s with the goal of replacing XML
  - Simpler and more efficient data interchange
- Originally designed for data interchange between client and server in web applications



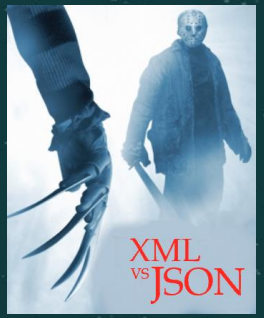
JSON
(JavaScript Object Notation)



- Based on the JS object syntax (*obviously*)
- Compatible with most modern programming languages.
- Currently very popular because:
  - **Lighter and simpler than XML.**
  - **Native JS support.**
- *De facto* standard in applications of all kinds.



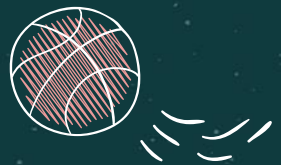
INTRODUCTION



XML	JSON
More detailed, but difficult to read	Simpler and more readable for humans
Tag-based	Uses {braces} and [brackets]
Heavier (because of tags)	Lighter (lacks tags)
Everything is text; data type settings come with validation	Supports data type natively (numbers, booleans, arrays...)
Very extensible, appropriate for structured documents	Less flexible, specifically designed for data interchange

SYNTAX

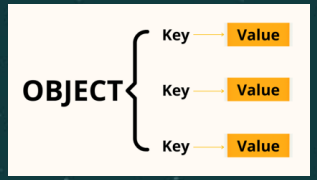
02.



SYNTAX

- Simple
- Mimics that of objects in JS

- Information stored in objects
- Objects contain data as key-value pairs (**properties**)



JavaScrip Object Notation (JSON)

key:
label for the value

value:
- data to be represented and exchanged
- multiple **data types**: string, number, boolean, array, object...

.json files

SYNTAX

- JSON objects are delimited by braces { }
- Properties within an object are separated by commas ,
- Key and value are separated by a colon :
- Keys are delimited by double quotes " " and are case-sensitive
- Value writing follows JS rules:
  - Text strings are written in double quotes " "
  - Numbers, as is; decimal point
  - Booleans, as is
  - Arrays , delimited by Brackets [ ]
    - elements contained are comma-separated ,
  - Key without a value → value is `null`

```
{
    "name": "Rick",
    "surname": "Sanchez",
    "age": 70,
    "alive": true,
    "inventions": ["Portal gun",
                   "Mind reading helmet"],
    "fav_drink": "vodka"
}
```

SYNTAX

example

```
{
    "name": "Chris",
    "age": 24,
    "email": "chris@example.org",
    "active": true,
    "subjects": [
        "LM",
        "ISO",
        "FHW",
        "PAR",
        "IPE1",
        "GBD"
    ],
    "address": {
        "streetname": "Darklong",
        "town": "Valencia",
        "zip": 46001
    }
}
```

objects

The value can also be another JSON object (nesting).

SYNTAX

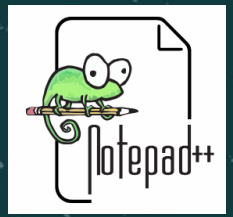
- Like XML, JSON **does not need spaces, tabs or line breaks**
  - But they are recommended to increase readability
- Special characters in strings must be escaped by prefixing \
  - Double quote → **\"**
  - Backslash → ****
  - Line break → **\n**
  - Tab → **\t**
- Numbers must not carry leading zeroes
- Booleans do not use quotation marks and are lowercase
- Arrays can contain any data types, even mixed

```
{
    "litany": "\t\"I must not fear.\nFear is
the mind-killer.\nFear is the little-death
that brings total obliteration.\nI will face
my fear.\nI will permit it to pass over me and
through me.\nAnd when it has gone past, I will
turn the inner eye to see its path.\nWhere the
fear has gone there will be nothing. Only I
will remain.",
    "book_info": [
        "Dune",
        "Frank Herbert",
        1965,
        false
    ]
}
```

SYNTAX

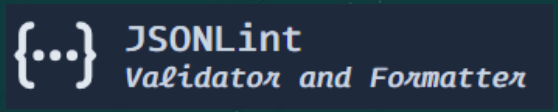
Recommended software for editing, checking syntax and formatting JSON

Plain text editor with JSON support



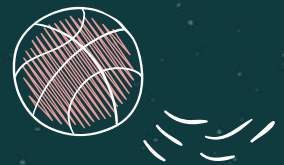
Any free IDE

Online JSON editor



VALIDATION

03.



VALIDATION

JSON Schema

- JSON documents can be validated to ensure well-formedness and appropriate data types.
  - JSON Schema was born with this goal
- Most programming languages have tools to convert data to JSON, so, JSON Schema is not very widespread.

- Its basics can be learnt by following the JSON Schema tour:

VALIDATION

JSON Schema

- JSON Schema validation can be done:
  - with the right plugins for plain text editors and IDEs: Notepad++, VS Code...
  - by programming, using tools and libraries available for many programming languages; for example: `ajv` (*Another JSON Schema Validator*) library for JavaScript.
  - Using an online tool, such as JSONSchema.dev
- The extension of JSON Schema files is `.schema.json`

VALIDATION

BASICS

- JSON Schema is written in JSON code
- The primitive data types are:

`string`	`integer`	`array`
`boolean`	`number`	`object`
	`null`	

- The main object definition starts with `"type": "object"`
- Then, the key names are declared inside `"properties"`
- Expected value data `"type"` must be declared within braces.

instance

```
{
  "species": "bull",
  "vertebrate": true
}
```

schema

```
{
  "type": "object",
  "properties": {
    "species": {
      "type": "string"
    },
    "vertebrate": {
      "type": "boolean"
    }
  }
}
```

VALIDATION

BASICS

- To validate nested objects (*objects within objects*), subschemas (*schemas within schemas*) are required.

instance

```
{
  "name": {
    "scientific": "Olea europea",
    "common": "Olive"
  },
  "max_height": 15,
}
```

schema

```
{
  "type": "object",
  "properties": {
    "name": {
      "type": "object",
      "properties": {
        "scientific": {
          "type": "string"
        },
        "common": {
          "type": "string"
        }
      }
    },
    "max_height": {
      "type": "integer"
    }
  }
}
```

VALIDATION

BASICS

- All properties are optional by default.
- To set obligation:

```
"required": ["key1", "key2", …]
```

instance

```
{
  "name": {
    "scientific": "Olea europea",
    "common": "Olive"
  },
  "max_height": 15,
}
```

- After closing the `"properties"` brace

schema

```
{
  "type": "object",
  "properties": {
    "name": {
      "type": "object",
      "properties": {
        "scientific": {
          "type": "string"
        },
        "common": {
          "type": "string"
        }
      },
      "required": ["scientific"]
    },
    "max_height": {
      "type": "integer"
    }
  }
}
```

VALIDATION

BASICS

- When a property has limited possible values → `enum`

  `"enum": ["value1", "value2", …]`

- `enum` replaces `type` keyword

instance

```
{
  "name": "traffic lights",
  "class": "electrical",
  "lit_color": "red"
}
```

schema

```
{
  "type": "object",
  "properties": {
    "name": {
      "type": "string"
    },
    "class": {
      "type": "string"
    },
    "lit_color": {
      "enum": ["red","amber","green"]
    }
  },
  "required": ["lit_color"]
}
```

VALIDATION

BASICS

- Arrays allow properties to store multiple values
- When an **`array`** is declared, the data types of its items must be declared → **`items`**
- The items in an array could be objects

instance

```
{
  "iban": "ES1122223333444455556666",
  "holders": ["12345678K", "88887946R"]
}
```

schema

```
{
  "type": "object",
  "properties": {
    "iban": {
      "type": "string"
    },
    "holders": {
      "type": "array",
      "items": {
        "type": "string"
      }
    }
  }
}
```

VALIDATION

RESTRICTIONS

- `string` values can have a limited length
  - `minLength`, `maxLength`
- Rules on string formation can also be set (*RegEx* `pattern`)
  - `[a-b]` → range of values (from `a` to `b`)
  - `{x}` → x characters from the range

    `+` → 1 or more `*` → 0 or more

instance

```
{
  "iban": "ES1122223333444455556666",
  "titulares": ["12345678K", "88887946R"]
}
```

schema

```
{
  "type": "object",
  "properties": {
    "iban": {
      "type": "string",
      "minLength": 24,
      "maxLength": 24,
      "pattern": "^ES[0-9]{22}$"
    },
    "titulares": {
      "type": "array",
      "items": {
        "type": "string",
        "maxLength": 9,
        "pattern": "^[0-9]+[A-Z]$"
      }
    }
  }
}
```

Pattern delimiters:
^ → opening
$ → ending

VALIDATION

RESTRICTIONS

- `number` and `integer` values can also be restricted:
  - `minimum`, `maximum`
  - `exclusiveMinimum`, `exclusiveMaximum`
  - `multipleOf`

instance

```
{
  "month": "november",
  "month_number": 11,
  "amount": 365.99
}
```

schema

```
{
  "type": "object",
  "properties": {
    "month": {
      "enum": ["january","february", //rest
      ]
    },
    "month_number": {
      "type": "integer",
      "minimum": 1,
      "exclusiveMaximum": 13
    },
    "amount": {
      "type": "number",
      "multipleOf": 0.01
    }
  }
}
```

VALIDATION

RESTRICTIONS

- Values can also be fixed (constant) → `const`

- Properties can be set to accept values from different data types:

  `"type": ["datatype1", "datatype2", …]`

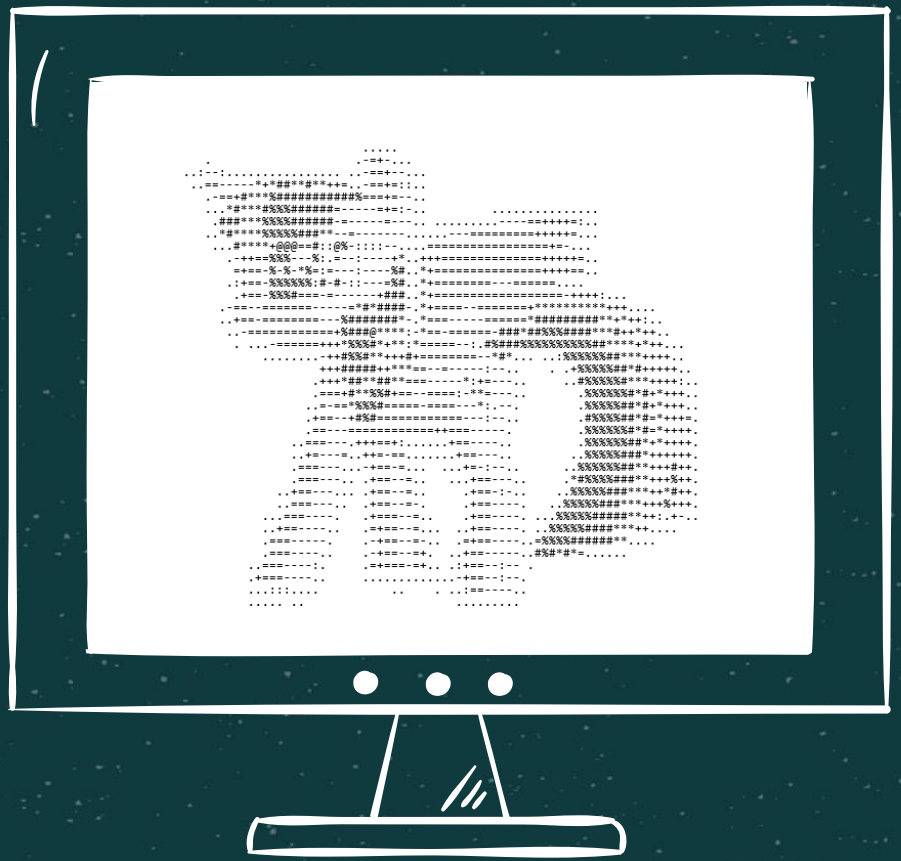
instance

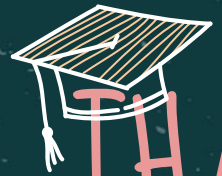
```
{
  "sender": "myself@email.com",
  "subject": null
}
```

schema

```
{
  "type": "object",
  "properties": {
    "sender": {
      "const": "myself@email.com"
    },
    "subject": {
      "type": ["string","null"]
    }
  }
}
```

PRÁCTICA 4.1





THANKS!

Do you have any questions?

g.domingomartinez@edu.gva.com