JSON

Oops. Wrong one!

2

2

JSON

- Javascript Object Notation
- For data representation and transmission
- .json
- Text based
- Minimal and portable
- Often used between web app and server
- Extension of Javascript – but no coding!
- Based on conventions seen in many languages
- Code for parsing JSON is available in many languages
- Comprised of key - value pairs

3

3

SIMPLE JSON OBJECT

```
{
   "first name" : "Julie",
   "last name"  : "Smith",
   "course"     : "Web Programming",
   "grade"      : 92
}
```

Notes:

- "key" : value
- Start and end with { }
- Keys will always be in quotes
- Values are strings, numbers, booleans, and null
  - or an array or other object containing these types
- Commas between pairs
- Validators exist to check your JSON
  https://jsonlint.com/

4

4

TRY IT!

- Create a JSON that will be used by an animal adoption center to represent its animals.
- The following data should be included:
  - Name
  - Type (ex, dog, cat, rabbit)
  - Breed
  - Age
  - Gender
- Use a validator to check your file

5

5

A VALUE CAN BE AN ARRAY

- Use [ ] notation to indicate an array

```
{
   "first name" : "Julie",
   "last name"  : "Smith",
   "course"     : "Web Programming",
   "grades"     :  [88, 95, 91, 92]
}
```


6

TRY IT!

- Add the following to your pet object
  - Medical procedures completed

7

NESTING OBJECTS

- A JSON object can contain other objects
- Example: The student name field can be a sub-class

```
{
   "name" : {
         "first"  : "Julie",
         "last"   : "Smith"
   }
   "course"  : "Web Programming",
   "grades"  :  [88, 95, 91, 92]
}
```


8

TRY IT!

- Add an "adopting family" field
- This should include
  - Name
  - Town
  - Number children in household
  - Other pets (true or false)

9

SET OF JSON OBJECTS

- You can create a group / array of objects using the [ ] notation

```
[
   {"id" : 1,  "type" : "rose"},
   {"id" : 2,  "type" : "carnation"},
   {"id" : 3,  "type" : "sunflower"}
]
```

10

10

TRY IT!

- Create a json file with three pets

11

11

JSON VS XML

- They have a similar purpose – transmission of data
- JSON is often considered easier and faster to write
- JSON is easier when you want to have an array
- XML has Schema as well as other technologies
- JSON has built-in support within Javascript as well as other languages
- The concept of name/value pairs is well understood – another vote for JSON

12

12

WAIT! THERE IS A SPEC FOR JSON SCHEMA

```
{
  "$schema": "http://json-schema.org/draft-04/schema#",
  "$id": "http://mysite.com/student.schema.json",
  "title": "Student",
  "description": "One student's data",
  "type": "object",
  "properties": {
    "first name": {
      "description": "Student first name",
      "type": "string"
    },
    "last name": {
      "description": "Student last name",
      "type": "string"
    },
    "course":{
        "description":"enrolled course",
        "type": "string"
    },
   "grade":{
        "description":"Grade average",
        "type":"integer"
    }
}
}
```

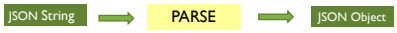
13

13

PARSING & SERIALIZATION: PARSE() & STRINGIFY()

- **stringify**() : serializes a JSON object
  - Turns a JSON into a JSON string that can be transferred
  - Works for Javascript objects as well

- **parse**() : parses a JSON string
  - Turns a JSON serialized string back into a JSON object



14

14

EXAMPLE

- student = {
  "name" :"Suzie",
  "course" :"comp20"
  }
- strStudent = JSON.stringify(student); // result is: {"name" :"Suzie","course" :"comp20" }
- objStudent = JSON.parse(strStudent); //restores to an object

15

15

toJSON

- toJSON tells stringify what to do
- Add it as a method to the JSON
- student = {
  "name" :"Suzie",
  "course" :"comp20",
  toJSON() { return this.name + ":" + this.course}
  }

16

16

AJAX: ASYNCHRONOUS OPERATIONS WITHIN JAVASCRIPT

- Asynchronous JavaScript and XML
- Fetches data asynchronously from a web server without needing to refresh the page

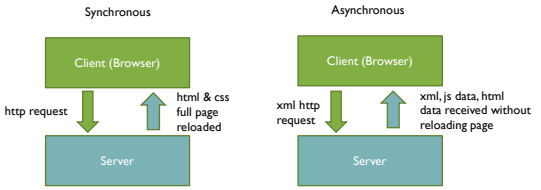
Gets data from server without needing to reload the page

Sends data in the background without needing to wait for a response

17

17

CONCEPTUAL VIEW



18

18

AJAX ELEMENTS

- **XMLHttpRequest** can send and receive data from web server.
- **readystate** has a value between 0 to 4 to indicate the status of request.
- **onreadystatechange** an event for the XMLHttpRequest object that is triggered when there is a change in the readystate value
- **open()/send()** methods of the XMLHttpRequest object to send the request
- Data can be transported using JSON or XML

19

19

READY STATE VALUES

0 : Unsent → nothing happened yet - open()not called
1 : Opened → send() not yet called
2 : Headers Received → send() and open() called
3 : Loading → Data is being received
4 : Done → Operation completed

20

20

COMMON HTTP STATUS CODES

200 → Success
201 → Resource was created
204 → Request is successful, but no data received.
404 → Page Not Found

21

21

OPEN() AND SEND()

req = new XMLHttpRequest();

req.open("post","data.php",true);

- Parameters:
  - post or get
  - Address of processing file on server (relative path)
  - Boolean: is this to be sent asynchronously (normally, true)
- req.send("id:101");
  - Uses a JSON string

22

22

PUTTING IT ALL TOGETHER

```
function requestData() {
  var reqObj = new XMLHttpRequest();
  if (! reqObj)
   {alert("Unable to create HTTPRequest object"); return;}
  data = "id:101";
  reqObj.onreadystatechange = getMyData();
  reqObj.open("POST", "getData.php", true);
  reqObj.send(data);
}
```

23

23

EXAMPLE, CONTINUED

```
function getMyData()
{
   if(this.readyState==4 && this.status==200)
         var data=this.responseText;
         var info=JSON.parse(data);
         for(i in info ){
            document.write(i + ":"+ info[i]);
         }
}
```

24

24

AJAX CHEAT SHEET

- On the HTML page:
  - Create div sections for information display as needed
- Initialize page
  - Set up any event handlers
- Create Request function
  - Create the HttpRequest object using several class types since you don't know in advance what will work on the client system
- Get Info function
  - Create a request object
  - Form the url for the request
  - Open the request
  - Set up the onreadystatechange event to the callback that processes the asynchronous info
  - Send the request
- Display Info function
  - Check the request ready state - a value of 4 means it is done
  - Check the request status – a value of 200 means completed successfully
  - The responseText property contains the information that came back from the server

25

CROSS-ORIGIN REQUEST SHARING (CORS)

- Security policy that applies when your browser fetches assets for a web page
  - Fonts
  - Images
  - Scripts
- Security policies minimize the risks associated with code that can hack a browser
  - Downloading malicious code
  - "Hijacking" the browser
  - Adding undesirable plugins

26

26

SECURITY POLICIES

- Same origin
  - "Documents" must have the same origin
  - A page hosted on a server can only interact with other documents that are also on that server
  - Even a different protocol (http vs. https) will be deemed as a different origin
- CORS
  - Cross origin requests are allowed
  - Server has to specify what can gain access and how they gain access
  - Accomplished with http headers: Access-Control-Allow-Origin
  - Headers can be set up on server or in .htaccess file:

```
<Files "*.json">
  Header set Access-Control-Allow-Origin "*"
</Files>
```

27

27

USING AJAX TO READ A JSON FILE

- Use the usual AJAX data pattern
- Use GET for data retrieval
- JSON file is the target of the request
- Callback function handles data retrieval

28

28