|  |
| --- |
| [[https://myetudes.org/etudes-melete-tool/images/printer.png](https://myetudes.org/portal/tool/4c4d3792-8b10-40ce-8016-d7a5ac569a1c/print_module.jsf?printModuleId=1436385323) Send to Printer](https://myetudes.org/portal/tool/4c4d3792-8b10-40ce-8016-d7a5ac569a1c/print_module.jsf?printModuleId=1436385323) | [Close Window](https://myetudes.org/portal/tool/4c4d3792-8b10-40ce-8016-d7a5ac569a1c/print_module.jsf?printModuleId=1436385323) |
|  | |
| 12. JSON  12.1. What is JSON?  *Copyright (c) 2014, Rula Khayrallah*  JSON stands for JavaScript Object Notation.  It is a**plain text data interchange format**. It is used in the transmission of data between machines.  JSON is easy for humans to read and write. It is easy for machines to parse and generate. It is based on a subset of JavaScript but it is completely language independent.   JSON is built on two structures:   * A collection of name/value pairs similar to a JavaScript object. * An ordered list of values similar to a JavaScript array.   In addition to arrays and objects, JSON supports strings, finite numbers, true, false, and null values.  The value undefined as well as the numbers NaN and Infinity are not representable in JSON.  The following examples all illustrate valid JSON format:  { "name": "Fido", "hungry": false, "lonely": false, "ill": false }  { "name": "Wanda", "hungry": false, "ill" :false }  [ 100, 80, 90, 89, 99 ]  **JSON requires strings to be enclosed in double quotes.**"Fido" is valid but 'Fido' is not.  **JSON also requires property names to be be enclosed in double quotes.** (JavaScript does not).  { "name": "Fido", "hungry": false, "lonely": false, "ill": false } is valid JSON.  { name: "Fido", hungry: false, lonely: false, ill: false } is NOT valid JSON.  12.2. Object Serialization  *Copyright (c) 2014, Rula Khayrallah*  Object serialization is the process of converting an object to a string from which it can later be restored.  The function **JSON.stringify()**  may be used i**n ECMAScript 5** to serialize JavaScript objects and convert them to JSON format.  The **JSON.stringify()** **function converts an object, an array or a primitive value to a JSON formatted string**.  It only includes an object's own enumerable properties.  Usually, JSON.stringify is called with a single argument and returns the corresponding string.  **Examples:**  >>> JSON.stringify(6.3);  "6.3"  >>> JSON.stringify(false);  "false"  >>> var myDog = { name: 'Fido', hungry: false, lonely: false, ill: false };  >>> JSON.stringify( myDog );  "{"name":"Fido","hungry":false,"lonely":false,"ill":false}"  Note that the string 'Fido' has been enclosed in double quotes: "Fido".  Note also that all the JavaScript property names (name, hungry, lonely, ill) have also been enclosed in double quotes.  >>> var grades = [ 100, 80, 90, 89, 99 ];  >>> JSON.stringify(grades);  "[100,80,90,89,99]"  JSON.stringify() takes an optional **replacer** argument that may be one of two things:  a function that will replace values before converting to string.  an array that contains the names of properties to be included in the output.  Note that the name 'replacer' is misleading in this case because the array is acting more like a filter rather than a replacer.  We can also specify an optional **space** argument that provides an indentation string or the number of spaces to use for indentation in order to get a more readable output.  If we specify a number, JSON.stringify() will insert newlines and use the specified number of spaces to indent each level of output.  If we specify a non-empty string instead, JSON.stringify() will insert newlines and use that string to indent each level.  **Examples:**  In the example below we only want the name and the hungry properties to be included in the output so we specify the array ['name', 'hungry'] as our 'replacer' argument.  We also specify 3 as our space argument, to have each level indented by 3 spaces.  >>> JSON.stringify(myDog, ['name', 'hungry'], 3)  "{     "name": "Fido",     "hungry": false  }"  If we don’t want to specify a replacer but still want to include a space argument, we can use null for the replacer. We can also specify any string to be used for indentation:  >>> JSON.stringify(myDog, **null,  '--->'**);  "{  --->"name": "Fido",  --->"hungry": false,  --->"lonely": false,  --->"ill": false  }"  >>>var restaurant=[{name: 'Red Apple', price: 3, rating: 2, distance: 2},                                     {name: "Zoe's Place", price: 1, rating: 5, distance: 1},                                     {name: 'Yumm', price: 2, rating: 3, distance: 5},                                     {name: 'California Foods', price: 5, rating: 4, distance: 8}];  >>> JSON.stringify(restaurant, null, 2);  "[    {      "name": "Red Apple",      "price": 3,      "rating": 2,      "distance": 2    },    {      "name": "Zoe's Place",      "price": 1,      "rating": 5,      "distance": 1    },    {      "name": "Yumm",      "price": 2,      "rating": 3,      "distance": 5    },    {      "name": "California Foods",      "price": 5,      "rating": 4,      "distance": 8    }  ]"  12.3. Parsing JSON  *Copyright (c) 2014, Rula Khayrallah*  The function **JSON.parse()** may be used i**n ECMAScript 5** to parse JSON formatted strings.  The JSON.parse() function returns the object, array, or primitive value that the JSON string represents.  **Examples:**  >>> JSON.parse("-8.7");  -8.7  >>> JSON.parse("true");  true  >>> JSON.parse("[ 100, 87, 90 ]");  [ 100, 87, 90 ]  >>> var myString = '{"name":"Alice","phone":"555-1234"}'  >>> JSON.parse(myString);  Object { name="Alice", phone="555-1234"}  JSON.parse() takes an optional **reviver** argument that will transform the parsed value before it is returned.  For example the function standardize() below may be used to standardize the value of the country property after the parsing:  function standardize( propName, value) {      // Standardize the country property      if (propName === "country") {          return "USA";      };    // Leave other properties unchanged      return value;  };  >>> var myString = '{"name":"Alice", "country":"United States", "phone":"555-1234"}';  >>>JSON.parse(myString, standardize);  Object { name="Alice", country="USA", phone="555-1234"}  12.4. Deep Copy with JSON  *Copyright (c) 2014, Rula Khayrallah*  To make a deep copy of an array (or any object) in JavaScript, we can simply convert it to JSON format and then parse it back.  Going back to our shallow copy example from module 7.9:  >>>var charlie = [ 100, 98, [85,90], 85 ];  >>>var diana = JSON.parse(JSON.stringify(charlie));  >>> diana   [100, 98, [85, 90], 85]  >>> charlie[1] = 0;  >>> charlie[2][1]=0;  >>> charlie  [100, **0**, [85, **0**], 85]  >>> diana   [100, **98**, [85, **90**], 85]  Or we can write a function to perform that:  // Make a deep copy of an array  function deepcopy(a) {              return JSON.parse( JSON.stringify(a));  }  12.5. JSON vs XML  *Copyright (c) 2014, Rula Khayrallah*  XML is another plain text format used to structure, store, and transfer data.  XML is based on a tree structure, and uses tags to delimit elements (like HTML).  However the tags are user-defined.  So for example to represent the information included in the JSON format:  {"name":"Alice", "country":"United States", "phone":"555-1234"}  We could use the following in XML:  <person>              <name>Alice</name>              <country>United States</country>              <phone>555-1234</phone>  </person>  Both JSON and XML use plain text and both formats are quite readable.  Both support Unicode strings and that translates into support for any language.  In addition, both use hierarchical structures.  **JSON results in shorter encodings for the same data**, mainly because it does not require any closing tags.  JSON also supports arrays.  Above all, JSON is easier to parse and generate from within JavaScript as well as from many other programming languages:  the reason is that it matches the native data structures of most programming  languages.  For AJAX applications, JSON has become the preferred data interchange format. |  |