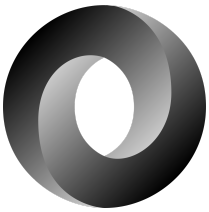
7.10 JS Data Structures

rpg map - answered by Rendell Soberano

The JSON object literal



- **JSON object literals** are a lightweight data interchange format in JavaScript, representing structured data as a collection of key-value pairs enclosed in curly braces.
  - They are suitable for easy data exchange and storage.
- JSON object literals are surrounded by curly braces {}.
  - JSON object literals contains key/value pairs.
- Keys and values are separated by a colon.

```
myObj = {"name":"John", "age":30, "car":null};
```

- You create a JavaScript object by parsing a JSON string.

```
myJSON = '{"name":"John", "age":30, "car":null}';
myObj = JSON.parse(myJSON);
```

Accessing Object values

- You can access object values by using dot (.) notation:

```
const myJSON = '{"name":"John", "age":30, "car":null}';
const myObj = JSON.parse(myJSON);
x = myObj.name;
```

Looping an Object

- You can loop through object properties with a for-in loop.

```
const myJSON = '{"name":"John", "age":30, "car":null}';
const myObj = JSON.parse(myJSON);

let text = "";
for (const x in myObj) {
  text += x + ", ";
}
```

Keyword(s)	Description	Sample code	Console output

`parse`	• Used to convert a JSON-formatted string into a JavaScript object.	`const fruit_json = '{"name": "Apple", "color": "Red", "taste": "Sweet"}';`
		`const fruit_obj = JSON.parse(fruit_json);`
		`console.log(fruit_obj);`
			`Object`
			`color: "Red"`
			`name: "Apple"`
			`taste: "Sweet"`

The Set data structure

- A JavaScript **Set** is a collection of unique values.
  - Each value can only occur once in a Set.
  - A Set can hold any value of any data type.
  - You cannot directly access elements by their index like you can with arrays because Sets are unordered collections

Keyword(s)	Description	Sample code	Console output

`add()`	• Adds an element to the set.	`const fruit_set = new Set();`
		`fruit_set.add("Apple");`
		`fruit_set.add("Banana");`
		`fruit_set.add("Orange");`
			`true`
			`true`

`has()`	• Checks if a specific element exists in		

	the set.	`console.log(fruit_set.has("Apple"));`
		
		`fruit_set.add("Grapes");`	

`delete()`	• Deletes a specified element from the set.	`const fruit_set = new Set();`
		`fruit_set.add("Apple");`
		`fruit_set.add("Banana");`
		`fruit_set.add("Orange");`
		
		`fruit_set.delete("Banana");`
		`console.log(fruit_set.has("Banana"));`
		
		`fruit_set.clear();`
		`console.log(fruit_set.has("Apple"));`
			`false`
			`false`

`clear()`	• `Removes all elements from the set.`		

`forEach()`	• Executes a function for each element in the set.	`const fruit_set = new Set();`
		`fruit_set.add("apple");`
		`fruit_set.add("banana");`
		`fruit_set.add("orange");`
		
		`console.log("Fruits in the Set:");`
		`fruit_set.forEach((fruit) => {`
		`  console.log(fruit);`
		`});`
			`apple`
			`banana`
			`orange`

`size`	• Returns the number of elements in the set.	`const fruits = new Set(["apple", "banana", "cherry", "date"]);`
		
		`const number_fruits = fruits.size;`
		
		`console.log("There are " + number_fruits + " fruits in the Set.");`
			`There are 4 fruits in the Set.`

The Map data structure

- A `Map` holds key-value pairs where the keys can be any datatype.
- It remembers the original insertion order of the keys.
- It has a property that represents the size of the map.
- You can create a Map by
  - Passing an Array to new `Map()`

    - Create a Map and use `Map.set()`
- You can create a Map by passing an Array to the new `Map()` constructor:

```
// Create a Map
const fruits = new Map([
  ["apples", 500],
  ["bananas", 300],
  ["oranges", 200]
]);
```

Keyword(s)	Description	Sample code	Console output

`set()`	• Used to add or update a key-value pair in the Map.	`const fruits = new Map();`
		
		`fruits.set("apples", 500);`
		`fruits.set("bananas", 300);`
		`fruits.set("oranges", 200);`
		
		`console.log(fruits.get("apples"));`
			`500`

`get()`	• Retrieves the value associated with a specific key in a Map.		

`clear()`	• Removes all key-value pairs, effectively emptying the Map.	`const fruits = new Map();`
		
		`fruits.set("apples", 500);`
		`fruits.set("bananas", 300);`
		`fruits.set("oranges", 200);`
		
		`console.log(fruits.get("apples"));`
		
		`fruits.delete("bananas");`
		`console.log(fruits.has("bananas"));`
		
		`fruits.clear();`
		`console.log(fruits.size);`
			`500`
			`false`
			`0`

`delete()`	• Removes a key-value pair from a Map based on the provided key.		
`size`	• Property of a Map object represents the number of key-value pairs in the Map.		
`has()`	• Checks whether a specific key exists in a Map.		

`forEach()`	• Iterates over key-value pairs in a Map and executes a	`const fruits = new Map();`
		
		`fruits.set("apples", 500);`
		`fruits.set("bananas", 300);`
			`Key: apples, Value: 500`
			`Key: bananas, Value: 300`

	provided function for each entry.	`fruits.set("oranges", 200);`
		
		`fruits.forEach((value, key) => {`
		`  console.log("Key: " + key + ", Value: " + value);`
		`});`
			`Key: oranges, Value: 200`

`entries()`	• Returns an iterable containing key-value pairs (as arrays) for all entries in a Map.	`const fruits = new Map();`
		
		`fruits.set("apples", 500);`
		`fruits.set("bananas", 300);`
		`fruits.set("oranges", 200);`
		
		`console.log("Keys:");`
		`for (const key of fruits.keys()) {`
		`  console.log(key);`
		`}`
		
		`console.log("Entries:");`
		`for (const [key, value] of fruits.entries()) {`
		`  console.log(`Key: ${key}, Value: ${value}`);`
		`}`
		
		`console.log("Values:");`
		`for (const value of fruits.values()) {`
		`  console.log(value);`
		`}`
			`Keys:`
			`apples`
			`bananas`
			`oranges`
			`Entries:`
			`Key: apples, Value: 500`
			`Key: bananas, Value: 300`
			`Key: oranges, Value: 200`
			`Values:`
			`500`
			`300`
			`200`

`keys()`	• Returns an iterable containing all the keys (identifiers) in a Map.		
`values()`	• Returns an iterable containing all the values in a Map.		

Additional Material

- **Learn more**
  - W3Schools