CHEAT SHEET FOR SPREAD SET, MAP, WEAKSET, WEAKMAP

JavaScript MAP

A Map holds key-value pair where key ca be any datatype. A Map remember order of the keys. A Map has a property that represents the size of the map.

➢ new Map():

The new Map() method is used to create new map.

Ex:
```
let fruits = new Map([ [“apple”, 20], [“banana”, 30] ]);
fruits.set([“mango”, 40]);
cosole.log(fruits);
```

O/P:
```
{  'apple' => 20, 'banana' => 30, 'mango' => 40 }
```

➢ .set() & .get() :

The set method is used to add element in array. The get() method can be used to get value from the map.

Ex:
```
let fruits = new Map([ [“apple”, 20], [“banana”, 30] ]);
fruits.set([“mango”, 40]);
let mangoQuatity = fruits.get(“mango”);
console.log(mangoQuatity);
```

O/P:
```
40
```

➢ .delete() & .clear() :

The delete() methid is used to delete element from the Map. While, clear() method clear whole Map.

Ex:
```
let fruits = new Map([ [“apple”, 20], [“banana”, 30], [“mango”,
40] ]);
console.log(fruits);
fruits.delete(“mango”);
console.log(fruits);
fruits.clear(); // Clear whole map
```

O/P:
```
{  'apple' => 20, 'banana' => 30, 'mango' => 40 }
{  'apple' => 20, 'banana' => 30 }
```

- ## .keys() & .values() :

  The keys() method return an iterator object of all keys in map. The values() method return an iterator object of all values in map.

```
Ex:   let fruits = new Map([ [“apple”, 20], [“banana”, 30], [“mango”,
      40] ]);
      console.log(fruits.keys());
      console.log(fruits.values());

O/P:  { “apple”, “banana”, “mango” }
      { 20, 30, 40 }
```

- ## .has() :

  The has() method check if map has defined key or not and return boolean value.

```
Ex:   let fruits = new Map([ [“apple”, 20], [“banana”, 30], [“mango”,
      40] ]);
      console.log(fruits.has(“apple”));
      console.log(fruits.has(“berry-berry”));

O/P:  true
      false
```

- ## .entries() :

  The entries() method returns an iterator object with [key, value] pair in a map.

```
Ex:   let fruits = new Map([ [“apple”, 20], [“banana”, 30], [“mango”,
      40] ]);
      console.log(fruits.entries());

O/P:  { [ 'apple', 20 ], [ 'banana', 30 ], [ 'mango', 40 ] }
```

- ## .forEach() :

  The forEach() method invokes a callback for each key/value pair in a map.

```
Ex:   let fruits = new Map([ [“apple”, 20], [“banana”, 30], [“mango”,
      40] ]);
      fruits.forEach( (value, key) => {
            console.log(value + “ - “ + key);
      })
```

```
O/P:  20 – apple
      30 – banana
      40 - mango
```

JavaScript SET

JavaScript Set is a collection of uniqe value. Each value occur in set only once. A Set can hold any value of any data type.

➢ new set() & .add():

The new Set() can be used to set value and add() method can be used to add element in Set.

```
Ex:   let char = new Set([“a”, “b”, “c”, “d”, “e”]);
      char.add(“f”);
      console.log(char)

O/P:  {  “a”, “b”, “c”, “d”, “e”, “f” }
```

➢ .delete() & .clear() :

The delete() method is used to delete element from the Set. While, clear() method clear whole Set.

```
Ex:   let char = new Set([ “a”, “b”, “c”, “d”, “e”, “f”]);
      char.delete(“f”);
      console.log(char);
      char.clear(); // Clear whole Set

O/P:  { “a”, “b”, “c”, “d”, “e” }
```

➢ .keys() & .values() :

The values() method return an iterator object of all values in set. The keys() method return same as values() because Set has no key.

```
Ex:   let char = new Set([ “a”, “b”, “c”, “d”, “e”, “f”]);
      console.log(char .keys());
      console.log(char .values());

O/P:  { “a”, “b”, “c”, “d”, “e”, “f” }
```

➢ .has() :

The has() method check if map has defined key or not and return boolean value.

```
Ex:    let char = new Set([ “a”, “b”, “c”, “d”, “e”, “f”]);
       console.log(char .has(“a”));
       console.log(char .has(“g”));

O/P:   true
       false
```

➢ .forEach() :

The forEach() method Invokes a callback for each element

```
Ex:    let char = new Set([“a”, “b”, “c”]);
       char.forEach( ele=> {
              console.log(ele);
       });

O/P:   a
       b
       c
```

➢ .entries() :

The entries() method returns an iterator object with [value, value] pair in a Set because Set has no key.

```
Ex:    const set1 = new Set();
       set1.add(42);
       set1.add(“45”);
       const iterator1 = set1.entries();
       for (const entry of iterator1) {
              console.log(entry);
       }

O/P:   [42, 42]
       [“45”, “45”]
```

JavaScript WeakMap

JavaScript WeakMap object is used to store the elements as key-value pair where keys are weakly referenced. WeakMap object can not contains the primitive type elements. It can contains only object type elements.

➢ new WeakMap():

The new WeakMap() method is used to create new weakMap in javascript.

Ex:
```
let weakmap = new WeakMap();
```

➢ .set() & .get():

The JavaScript WeakMap Set() method is used to add or update an element to WeakMap object with the particular key-value pair. Each value must have a unique key.

The JavaScript WeakMap get() method returns the value of specified key of an element from a WeakMap object.

Ex:
```
let weakmap = new WeakMap();
let obj1 = {}; let obj2 = {};
weakmap.set(obj1, “Object-1”);
weakmap.set(obj2, “Object-2”);
console.log(weakmap.get(obj1));
console.log(weakmap.get(obj1));
```

O/P:
```
Object-1
Object-2
```

➢ .delete():

The JavaScript WeakMap delete() ethod is used to remove the specified element from a WeakMap object.

Ex:
```
let weakmap = new WeakMap();
let obj1 = {};
weakmap.set(obj1, “Object-1”);
weakmap.delete(obj1);
console.log(weakmap.has(obj1));
```

O/P:
```
false
```

- ## .has() :

  The JavaScript WeakMap has() method indicates whether the WeakMap object contains the specified key. It returns true if the specified key is present, otherwise false.

  Ex:
  ```
  let weakmap = new WeakMap();
  let obj1 = {}; var obj2 = {};
  weakmap.set(obj1, “Object-1”);
  console.log(weakmap.has(obj1));
  console.log(weakmap.has(obj2));
  ```

  O/P:
  ```
  true
  false
  ```

JavaScript WeakSet

The JavaScript WeakSet object is the type of collection that allows us to store weakly held objects. Unlike Set, the WeakSet are the collections of objects only. It doesn't contain the arbitrary values.

- ## new WeakSet():

  The new WeakMap() method is used to create new weakMap in javascript.

  Ex:
  ```
  let weakset = new WeakSet();
  ```

- ## .add():

  The javascrpit WeakSet add() method can be used to add a new object to the end of WeakSet object.

  Ex:
  ```
  let weakSet = new WeakSet();
  let obj1 = {}; let obj2 = {};
  weakmap.add(obj1);
  weakmap.add(obj2);
  ```

- ## .has() :

The JavaScript WeakSet has() method indicates whether the WeakSet object contains the specified object. It returns true if the specified object is present, otherwise false.

Ex:
```
let weakset = new WeakSet();
let obj1 = {}; var obj2 = {};
weakset.add(obj1);
console.log(weakset .has(obj1));
console.log(weakset .has(obj2));
```

O/P:
```
true
false
```

- ## .delete():

The JavaScript WeakSet delete() method is used to remove the specified object from WeakSet object.

Ex:
```
let weakset = new WeakSet();
let obj1 = {};
weakset.add(obj1);
console.log(weakset .has(obj1));
weakset.delete(obj1);
console.log(weakset .has(obj1));
```

O/P:
```
true
false
```