Gaza Sky Geeks
Node JS Developer

Task 3

Iterable Protocol and Generators in JavaScript

Student: Alaa Sami Eid

Iterable Protocol

An Iterable is an Iterable Object

An Iterable can be iterated over with for of loops

Iterable: لو بدي احكي شو يعني Iterable (هو كائن قابل للتكرار).

The JavaScript for of statement loops through the elements of an iterable object:

طيب for of loops: يعني بالبداية هو نوع من أنواع loops يستخدم مع الكائنات القابلة للتكرار, بمشي عليه عنصر عنصر.

Syntax

for (variable of iterable) {

}

Symbol.iterator

- Symbol.iterator is a well-known built-in symbol in JavaScript.

الان for of بستخدمها علشان امرر العناصر وتكرار للاشي الي بدي إياه ب loop , وميثود (Symbol.iterator) بتكون موجودة كميثود في map , string , array ، set وبستخدم for of بشكل عادي , بس (Symbol.iterator) مش موجودة بال object بس انا ممكن اصنع التكرار iterator وانشأها واستخدمها واعمل ميثود لحتى اقدر امرر على العناصر

-This object is **iterable** it can be looped over using for of

(هذا الكائن يمكن تكراره (iterated) باستخدام for of)

Iterating

Iterating means looping over a sequence of elements.

Here are some easy examples:

- Iterating over a String

– Iterating over an Array

- Iterating over a Set

– Iterating over a Map

Iterating Over a String

You can use a for of loop to iterate over the elements of a string:

Example

```
> console.log("alaa eid")
  const name = "alaa";
  for (const x of name) {
  console.log(x)

  }
  alaa eid
  a
  l
2 a
< undefined
>
```

Iterating Over an Array

You can use a for of loop to iterate over the elements of an Array:

Example

```
> const letters = ["a","b","c"];
  for (const x of letters) {
    console.log(x)
  }
  a
  b
  c
< undefined
>
```

Iterating Over a Set

You can use a for of loop to iterate over the elements of a Set:

Example

```
> const letters = new Set(["a","b","c"]);
  for (const x of letters) {
    console.log(x)
  }
  a
  b
  c
< undefined
>
```

Iterating Over a Map

You can use a for of loop to iterate over the elements of a Map:

Example

```
> const fruits = new Map ([
    ["apples", 500],
    ["bananas", 300],
    ["oranges", 200]
  ]);
  for (const x of fruits) {
    console.log(x)
  }
  ▸ (2) ['apples', 500]
  ▸ (2) ['bananas', 300]
  ▸ (2) ['oranges', 200]
< undefined
>
```

JavaScript Iterators:

The **iterator protocol** defines how to produce a **sequence of values** from an object.

An object becomes an **iterator** when it implements a next () method.

The next() method must return an object with two properties:

value (the next value)

done (true or false)

هان لما استخدم next يعني iterator يعني لما استخدم next بتعطيني القيمة التالية

next() بترجع **value** and **done**

Value القيمة التالية , **done** الها حالتين اما true or false (لو كانت done = false يعني لسا فيه قيم ضايل , بس لو done = true يعني فش قيم ضايل)

هاد المثال بدون (Symbol.iterator and for of) بس ب nextوبنعمل iterator

هان استخدمت nextوحطيت شرط لما يصل لمرحلة معينة done = true وطول ما فيه قيم ومستمر خلي done == false

```
> function myNumbers() {
    let n = 0;
    return {
      next: function() {
        if (n >= 30) {
          return { done: true };
        }
        n += 10;
        return { value: n, done: false };
      }
    };
  }

  const n = myNumbers();
  console.log(n.next());
  console.log(n.next());
  console.log(n.next());
  console.log(n.next());
  ▸ {value: 10, done: false}
  ▸ {value: 20, done: false}
  ▸ {value: 30, done: false}
  ▸ {done: true}
< undefined
>
```

بس هان لما طبعت ضليت أقول next next علشان اجيب القيم

الحل حعمل مثال ثاني ب **Symbol.iterator** وحستخدم for of حتى اقدر امرر العناصر ب loop وهيك حيصير **Iterable**

هان استخدمت Symbol.iterator and for of

هان لما حطيت **Symbol.iterator** قدرت استخدم **for of** وحددت done متى تصير true وتوقف ومتى تكمل

في JavaScript ما بقدر اكرر و امرر على object باستخدام for...of، لكن لما أضيف Symbol.iterator، بصير تقدر

وهيك صار Mynumbers **(Iterable)**

```
> myNumbers = {};
  myNumbers[Symbol.iterator] = function() {
    let n = 0;
    let done = false;
    return {
      next() {
        n += 10;
        if (n > 100) {
          done = true;
          return { done: true };
        }
        return { value: n, done: false };
      }
    };
  };

  for (const num of myNumbers) {
    console.log(num);
  }
  10
  20
  30
  40
  50
  60
  70
  80
  90
  100
< undefined
>
```

Generator:-

The Generator object is returned by a generator function and it conforms to both the iterable protocol and the iterator protocol.

لما اعمل مناداة لدالة generator الي بكتبها ب function* ما بيطلع النتيجة مباشرة لا , هو بيطلع Generator object

طبعا Generator object هو يلي بتعامل معه لما بدي استخدم next or for of.

طبعا كمان Generator object بيطبق وبنفذ بداخله iterable and iterator

يعني بقدر أقول ب generator انو بقدر استخدم for of

generator مهم لتنظيم استخدام البيانات بدل ما نحمل كل البيانات دفعة وحدة ننتجها واحدة واحدة حسب الحاجة

Generator is a subclass of the hidden Iterator class.

الـ Generator object بيعتبر نوع خاص من الـIterator

يعني بيرث خصائص وسلوك ال Iterator بشكل مخفي يعني مش واضح بالكود

يعني كل Generator هوIterator ، بس أكثر تخصص وبيضيف ميزات مثل القدرة على التوقف والاستئناف yield

الـ Generator بيشتغل خطوة خطوة، وبكل مرة بيرجع لك قيمة باستخدام yield

هان generator نوع خاص من iterator يرث الميثود Symbol.iterator بشكل تلقائي وبصير بكل سهولة وبدون تعقيد استخدم التكرار وال for of لأنه قابل للتكرار(Iterable)

هان بقدر استخدم next

```
> function* myGenerator() {
    yield 1;
    yield 2;
    yield 3;
  }const gen = myGenerator();

  console.log(gen.next());
  console.log(gen.next());
  console.log(gen.next());
  console.log(gen.next());

  ▸ {value: 1, done: false}
  ▸ {value: 2, done: false}
  ▸ {value: 3, done: false}
  ▸ {value: undefined, done: true}
< undefined
```

وبقدر استخدم for of هان بشكل افضل

```
> for (let val of myGenerator()) {
    console.log(val);
  }

  1
  2
  3
< undefined
```

شو بيعمل yield : بستخدمها لما بدي ارجع قيمة بشكل مؤقت ولو بدي أوقف تنفيذ الدالة مؤقتا

يعني بتوقف تنفيذ الكود مؤقتا وبترجع القيمة يلي بعدها ولما تستدعي next مرة ثانية بتكمل من نفس المكان

(لما نكتب yield 5، هذا السطر **يرجع القيمة 5** وبيوقف تنفيذ الدالة مؤقتًا عنده ولما نستدعي ()next. مرة ثانية، **بكمل من بعد هاد السطر**، يعني من السطر اللي **بعد** yield 5).