



UMassAmherst

Manning College of Information
& Computer Sciences

Programming Methodology
Lab 7: Iterators

Wednesday October 15, 2025

Weekly Lab Agenda

- Go over reminders/goals
- Review past material
- Work in groups of 2-3 to solve a few exercises
 - Please sit with your group from last week.
- Discussion leaders will walk around and answer questions
- Solutions to exercises will be reviewed as a class
- Attendance taken at the end

Reminders

- HW 4b is due this Sunday (10/19) at 11:59pm.
 - Come to office hours for help!
- Midterm 1 regrade requests are open until next Monday (10/20).
 - Please read through any comments left on your midterm.
- If you need to miss lab and have a valid reason according to the syllabus (medical, other personal) please fill out the questionnaire on Canvas before the start time of your lab.
 - Waking up late, bus was late are NOT valid reasons to miss lab.

Today's Goals

- Iterators

Iterators Review

```
interface Iterator<ValueType> {  
    next(): IteratorResult<ValueType>;  
}
```

```
interface IteratorResult<ValueType> {  
    done: boolean; // true indicates Iterator has reached its end  
    value?: ValueType; // could be undefined, specifically if done is true  
}
```

These interfaces are built-in to JS/TS. Conceptually, an iterator is an object that provides access to items in some collection following some order.

In JS/TS, the `next()` method of an iterator yields the next item and moves the iterator forward. The result of `next()` is returned in the form of an `IteratorResult`, an object specifying the resulting item and whether the iterator has completed.

Iterators Review

```
interface Iterable<ValueType> {  
    [Symbol.iterator](): Iterator<ValueType>;  
}
```

```
interface IterableIterator<ValueType> {  
    next(): IteratorResult<ValueType>;  
    [Symbol.iterator](): Iterator<ValueType>;  
}
```

The `Iterable` interface is a built-in interface in JS/TS that describes objects that can be iterated through, i.e. using a range based (`for... of ...`) loop. The method `[Symbol.iterator]()` is a method with a special name that returns an iterator for the iterable.

The `IterableIterator` interface is built-in and is equivalent to implementing both the `Iterable` and `Iterator` interfaces (i.e. implements `Iterable<ValueType>`, `Iterator<ValueType>`).

Iterators Review

For example, the class to the right is an example of a class that follows the Iterator and Iterable interfaces.

The *IterableIterator* interface is equivalent to doing implements `Iterable<T>`, `Iterator<T>`.

Note: this iterator never terminates! ...and is also not very useful...

```
class MyIterator implements IterableIterator<number> {  
    private _num: number;  
  
    constructor(n: number) {  
        this._num = n;  
    }  
  
    next(): IteratorResult<number> {  
        return { done: false, value: this._num };  
    }  
  
    [Symbol.iterator]() {  
        return this;  
    }  
}
```

Exercise 1: Iterators

- Implement the function `makeRepeat` that returns an instance of a custom `RepeatIterator` class.
- The `RepeatIterator` class should be an iterable iterator that produces the given value `N` times. `N` is assumed to be a non-negative integer.
- Hint: you may need to implement some interfaces.

```
class RepeatIterator<T> {  
    // TODO: Implement this class  
}
```

```
function makeRepeat<T>(value: T, count: number) {  
    // TODO: Implement this function  
    return new RepeatIterator(); // replace as needed  
}
```


Exercise 2: More Iterators

- Write a function that takes in an array of iterables and returns an instance of a custom ChainIterator class.
- The ChainIterator class should be an iterable iterator that yields items from the first iterable until it runs out. It should then yield items from the next iterable in the given array, repeating this process until the end of the array.
- You may assume the given array is not empty.

```
class ChainIterator<T> {  
    // TODO: Implement this class  
}  
  
function makeChain<T>(iterables: Iterable<T>[]): ChainIterator<T> {  
    // TODO: Implement this function  
    return new ChainIterator(); // replace as needed  
}
```