

Java Lambdas Part 1 Notes

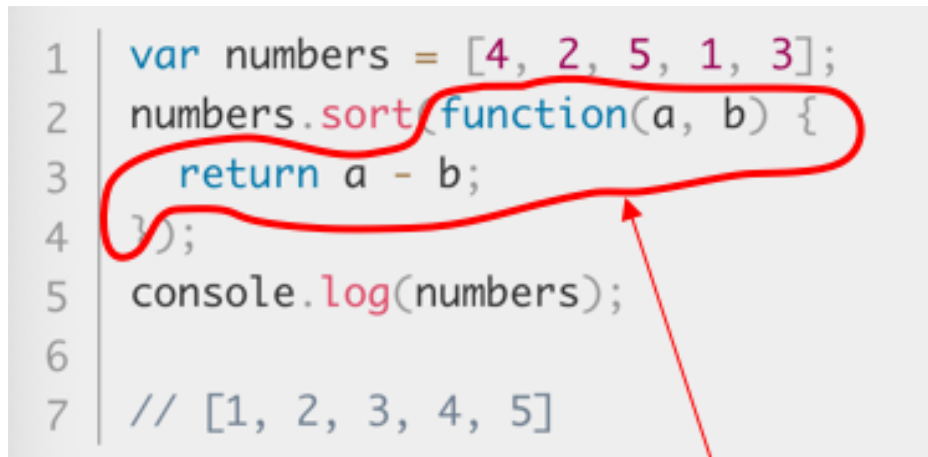
Team Treehouse

June 2, 2020

1 Old School

- Consider the sorting of two objects
- In javascript, this is achieved as follows

```
1 | var numbers = [4, 2, 5, 1, 3];  
2 | numbers.sort(function(a, b) {  
3 |     return a - b;  
4 | });  
5 | console.log(numbers);  
6 |  
7 | // [1, 2, 3, 4, 5]
```



This here :)

- In Java, to accomplish the above, used comparator interface with compare method
- Uses a lot of lines, an

```
import java.util.Collections;
import java.util.Comparator;
import java.util.List;

public class Main {

    public static void usingAnonymousInlineClass() {
        List<Book> books = Books.all();
        Collections.sort(books, new Comparator<Book>() {
            @Override
            public int compare(Book b1, Book b2) {
                return b1.getTitle().compareTo(b2.getTitle());
            }
        });

        public static void main(String[] args) {
            // write your code here
        }
    }
}
```

Comparator interface

Compare method

- Solution → Lambda

2 Introducing Lambdas

- Is syntactic sugar for interfaces
- Uses ‘->’

```
public static void usingLambdasInShortForm() {
    List<Book> books = Books.all();
    Collections.sort(books, (Book b1, Book b2) -> {
        return b1.getTitle().compareTo(b2.getTitle());
    });

    for (Book book : books) {
        System.out.println(book);
    }
}
```

This little guy here :)

- This is very similar to javascript’s ‘=>’

```
1 | let numbers = [4, 2, 5, 1, 3];  
2 | numbers.sort((a, b) => a - b);  
3 | console.log(numbers);  
4 |  
5 | // [1, 2, 3, 4, 5]
```



This guy here :)