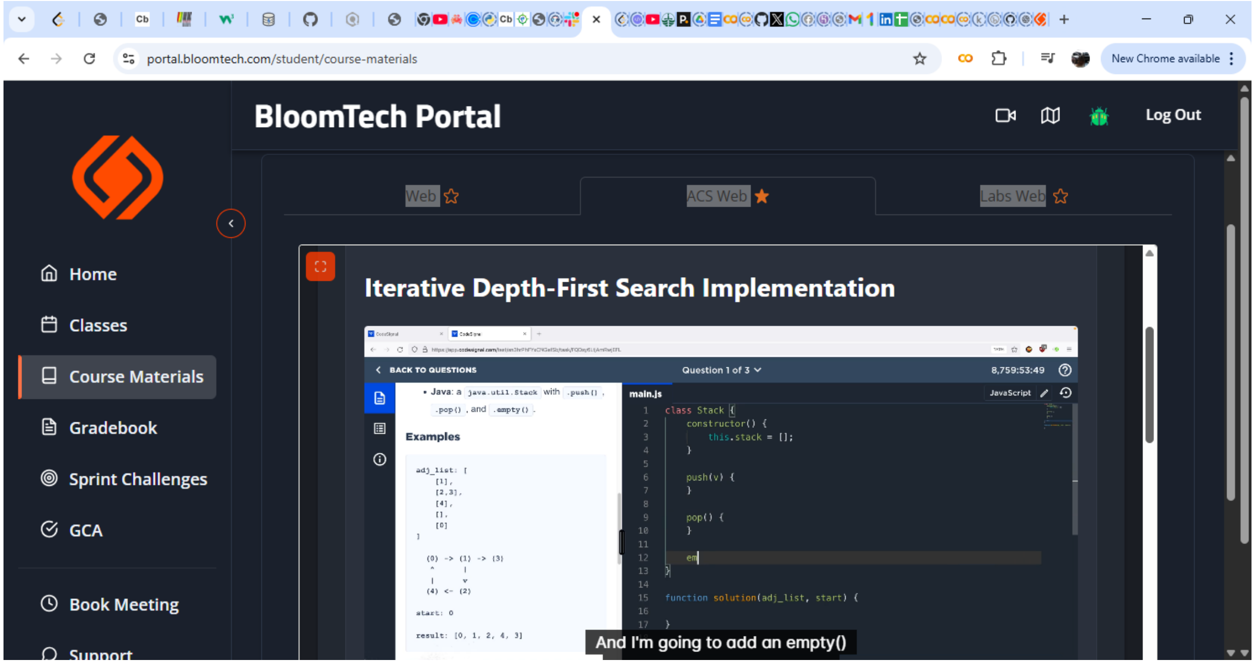


Iterative Depth-First Search Implementation

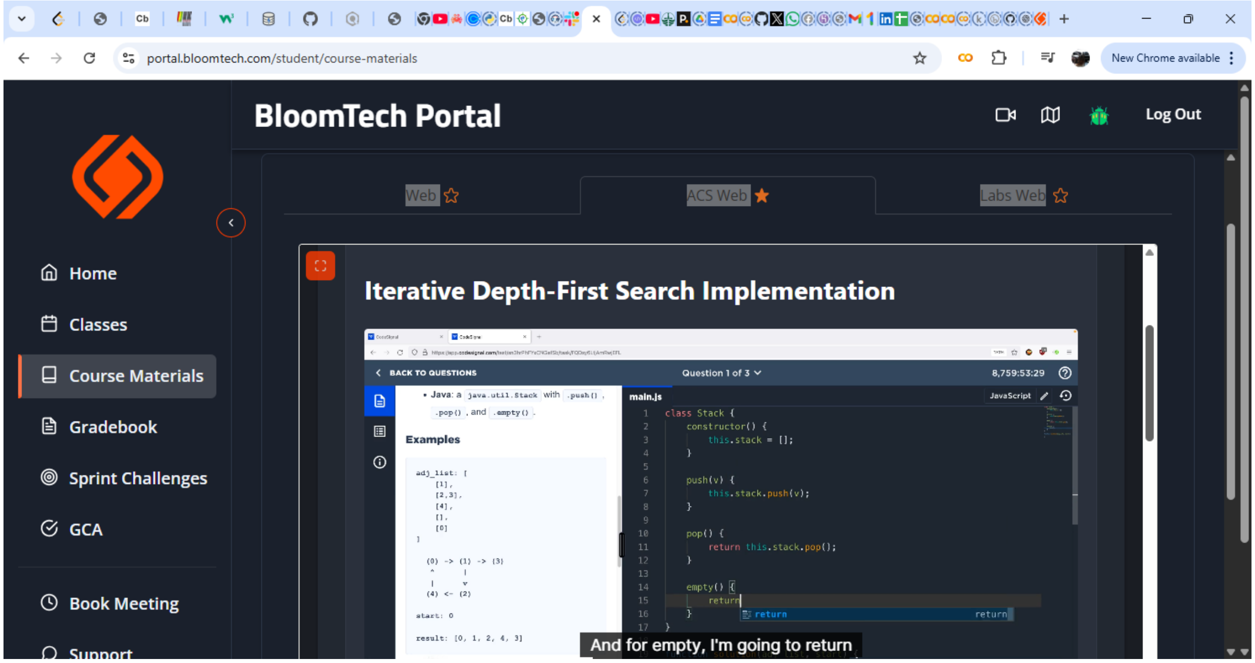
- Java: a `java.util.Stack` with `.push()`, `.pop()`, and `.empty()`.

Examples

```
adj_list: [
    [1],
    [2,3],
    [4],
    [],
    [0]
]

  (0) -> (1) -> (3)
   ^       |
   |       v
  (4) <- (2)

start: 0

result: [0, 1, 2, 4, 3]
```

```
class Stack {
    constructor() {
        this.stack = [];
    }

    push(v) {
    }

    pop() {
    }

    em
}

function solution(adj_list, start) {

}
```

And I'm going to add an empty()



Iterative Depth-First Search Implementation

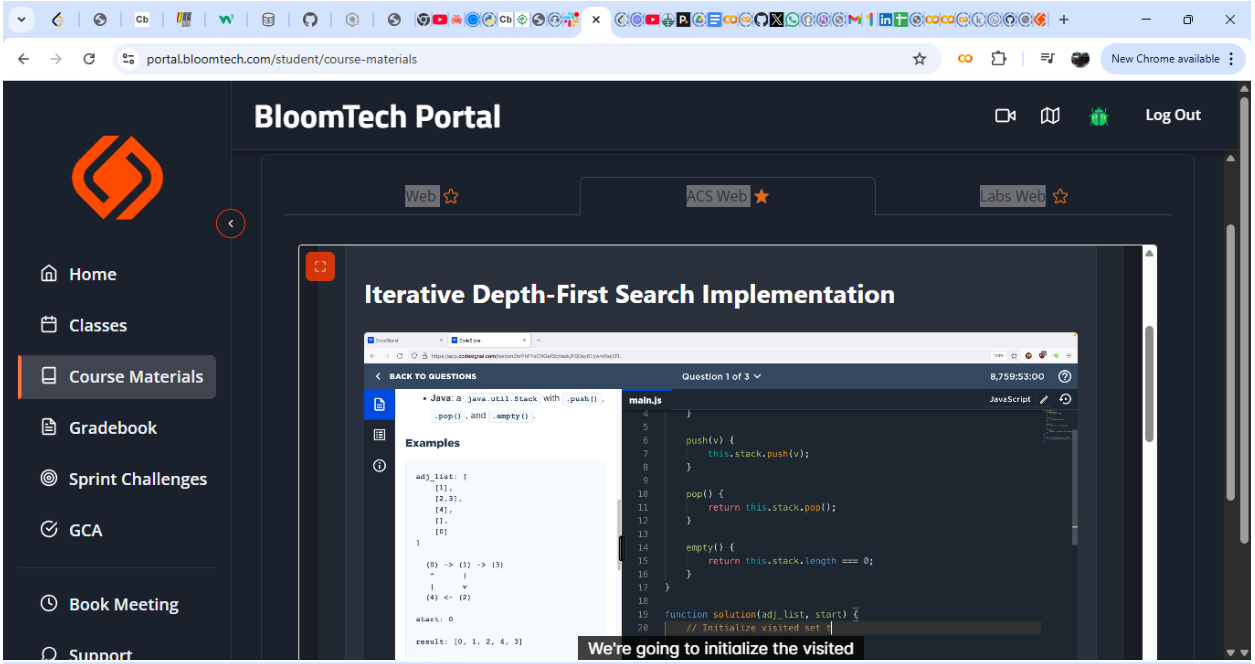
- Java: a `java.util.Stack` with `.push()`, `.pop()`, and `.empty()`.

Examples

```
adj_list: [
    [1],
    [2,3],
    [4],
    [],
    [0]
]

  (0) -> (1) -> (3)
   ^       |
   |       v
  (4) <- (2)

start: 0

result: [0, 1, 2, 4, 3]
```

```
class Stack {
    constructor() {
        this.stack = [];
    }

    push(v) {
        this.stack.push(v);
    }

    pop() {
        return this.stack.pop();
    }

    empty() {
        return
    }
}
```

And for empty, I'm going to return



Iterative Depth-First Search Implementation

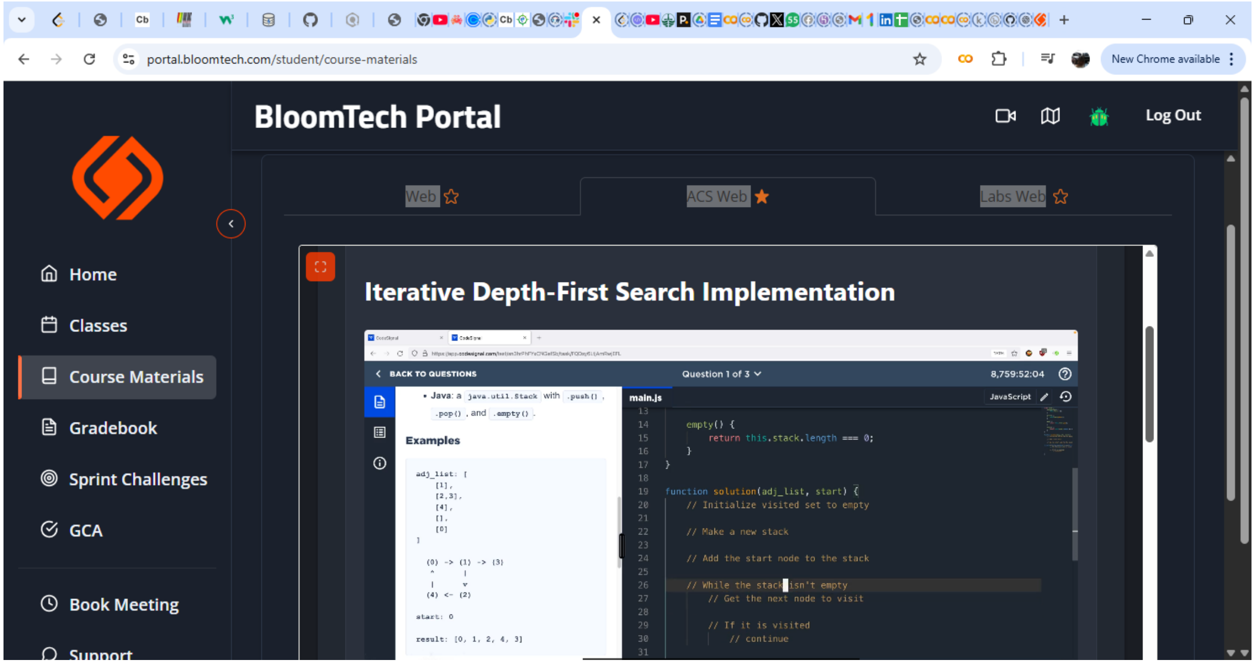
- Java: a `java.util.Stack` with `.push()`, `.pop()`, and `.empty()`.

Examples

```
adj_list: [
    [1],
    [2,3],
    [4],
    [],
    [0]
]

  (0) -> (1) -> (3)
   ^      |
   |      v
  (4) <- (2)

start: 0

result: [0, 1, 2, 4, 3]
```

```
    }

    push(v) {
        this.stack.push(v);
    }

    pop() {
        return this.stack.pop();
    }

    empty() {
        return this.stack.length === 0;
    }
}

function solution(adj_list, start) {
    // Initialize visited set t
```

We're going to initialize the visited



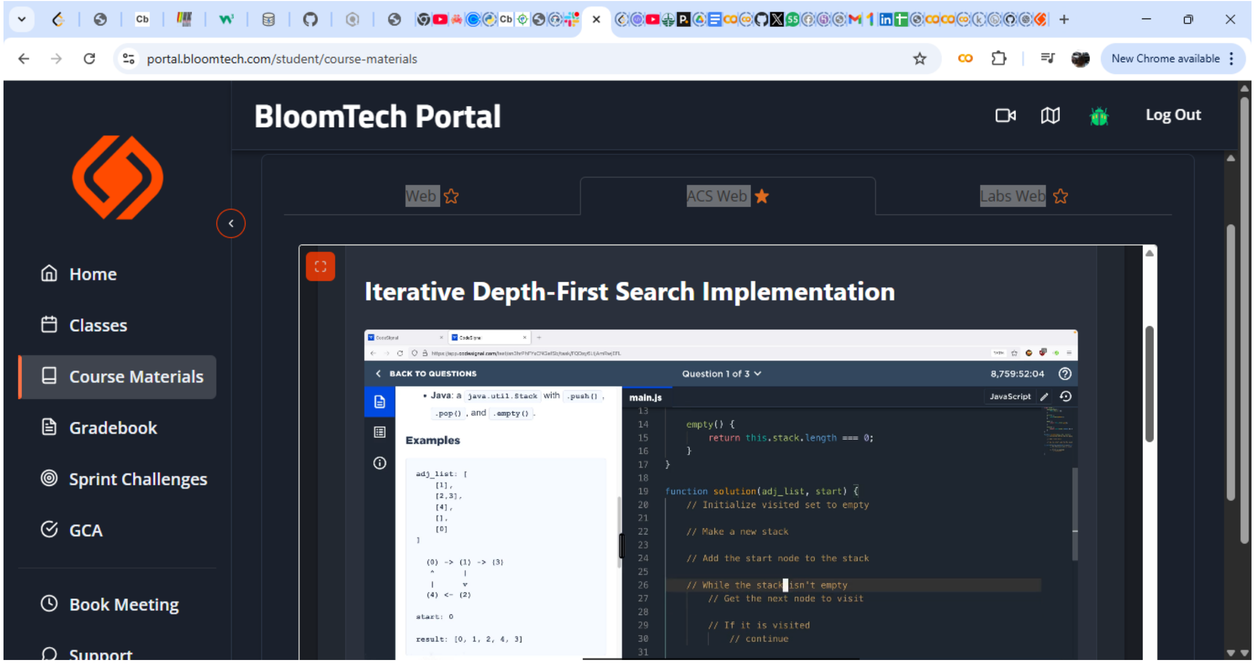
Iterative Depth-First Search Implementation

- Java: a `java.util.Stack` with `.push()`, `.pop()`, and `.empty()`.

Examples

```
adj_list: [
    [1],
    [2,3],
    [4],
    [],
    [0]
]

  (0) -> (1) -> (3)
   ^      |
   |      v
  (4) <- (2)

start: 0

result: [0, 1, 2, 4, 3]
```

```
    empty() {
        return this.stack.length === 0;
    }
}

function solution(adj_list, start) {
    // Initialize visited set to empty

    // Make a new stack

    // Add the start node to the stack

    // While the stack isn't empty
        // Get the next node to visit

        // If it is visited
            // continue

```



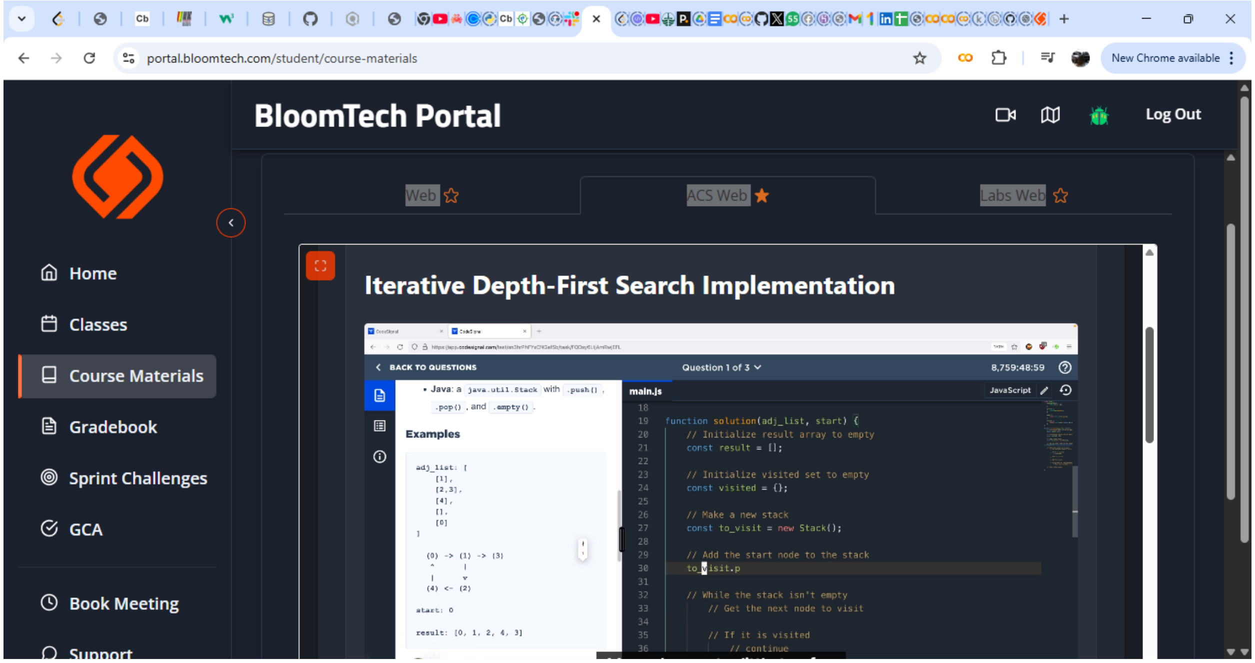
Iterative Depth-First Search Implementation

- Java: a `java.util.Stack` with `.push()`, `.pop()`, and `.empty()`.

Examples

```
adj_list: [
    [1],
    [2,3],
    [4],
    [],
    [0]
]

  (0) -> (1) -> (3)
   ^      |
   |      v
  (4) <- (2)

start: 0

result: [0, 1, 2, 4, 3]
```

```
    empty() {
        return this.stack.length === 0;
    }
}

function solution(adj_list, start) {
    // Initialize visited set to empty

    // Make a new stack

    // Add the start node to the stack

    // While the stack isn't empty
        // Get the next node to visit

        // If it is visited
            // continue

```



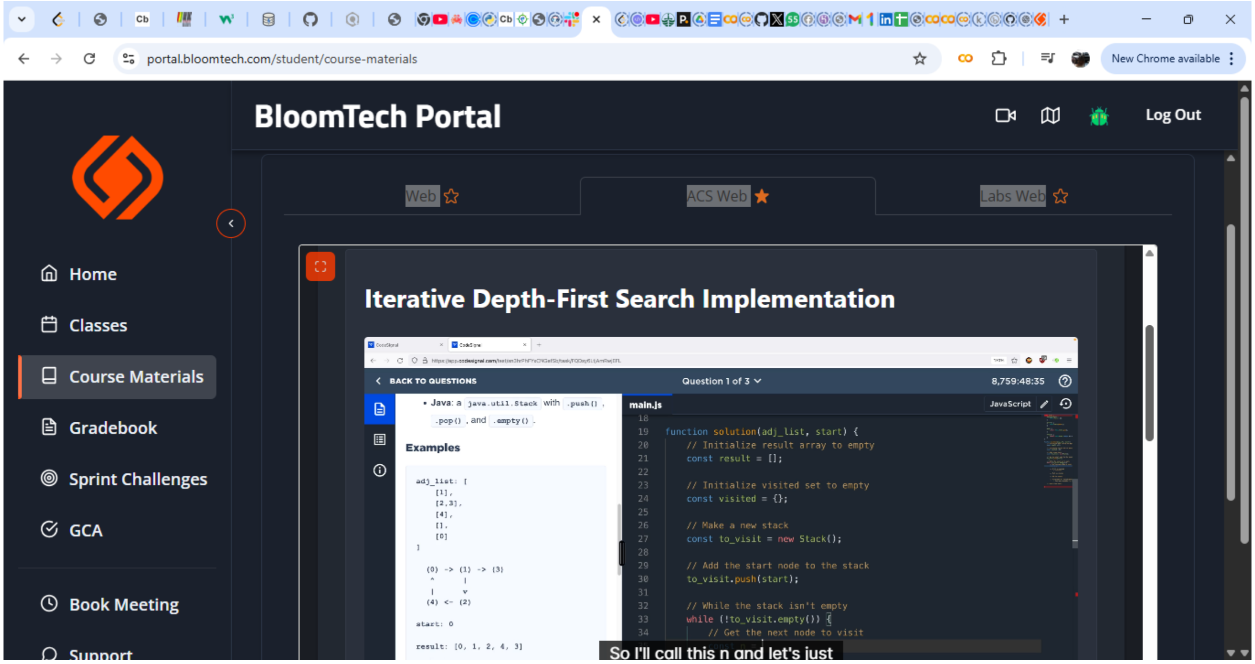
Iterative Depth-First Search Implementation

- Java: a `java.util.Stack` with `.push()`, `.pop()`, and `.empty()`.

Examples

```
adj_list: [
    [1],
    [2,3],
    [4],
    [],
    [0]
]

  (0) -> (1) -> (3)
   ^      |
   |      v
  (4) <- (2)

start: 0

result: [0, 1, 2, 4, 3]
```

```

function solution(adj_list, start) {
    // Initialize result array to empty
    const result = [];

    // Initialize visited set to empty
    const visited = {};

    // Make a new stack
    const to_visit = new Stack();

    // Add the start node to the stack
    to_visit.p

    // While the stack isn't empty
        // Get the next node to visit

        // If it is visited
            // continue
```



Iterative Depth-First Search Implementation

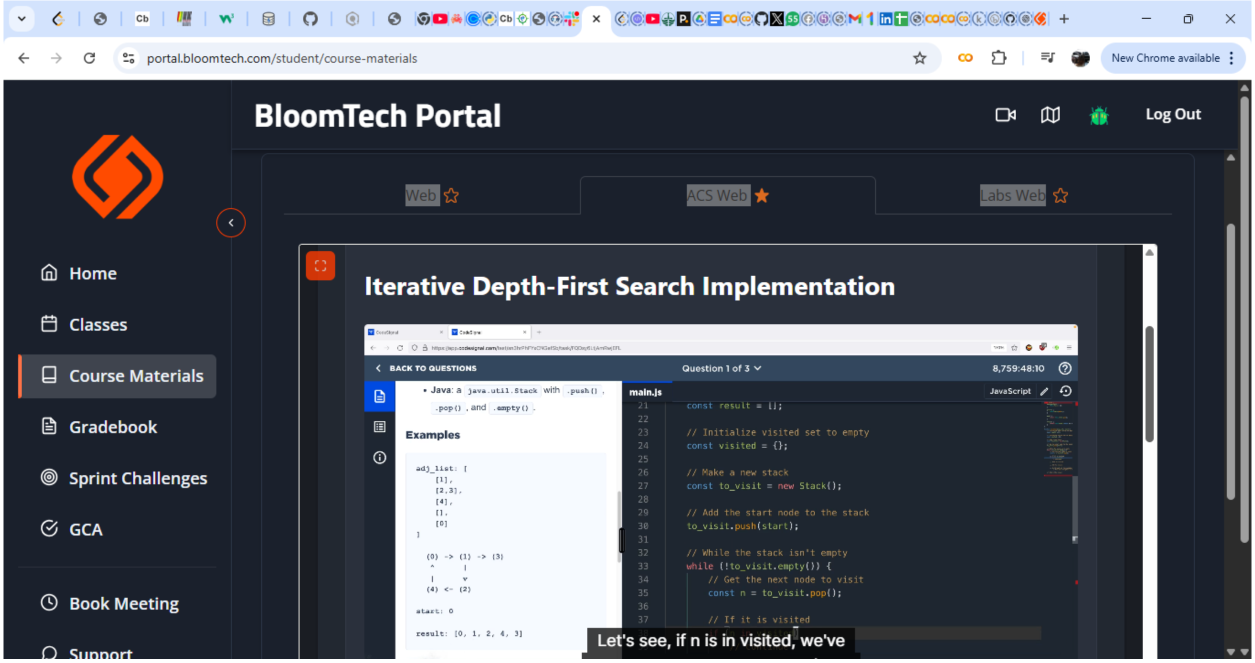
- Java: a `java.util.Stack` with `.push()`, `.pop()`, and `.empty()`.

Examples

```
adj_list: [
    [1],
    [2,3],
    [4],
    [],
    [0]
]

  (0) -> (1) -> (3)
   ^      |
   |      v
  (4) <- (2)

start: 0

result: [0, 1, 2, 4, 3]
```

```
function solution(adj_list, start) {
    // Initialize result array to empty
    const result = [];

    // Initialize visited set to empty
    const visited = {};

    // Make a new stack
    const to_visit = new Stack();

    // Add the start node to the stack
    to_visit.push(start);

    // While the stack isn't empty
    while (!to_visit.empty()) {
        // Get the next node to visit
```

So I'll call this n and let's just



Iterative Depth-First Search Implementation

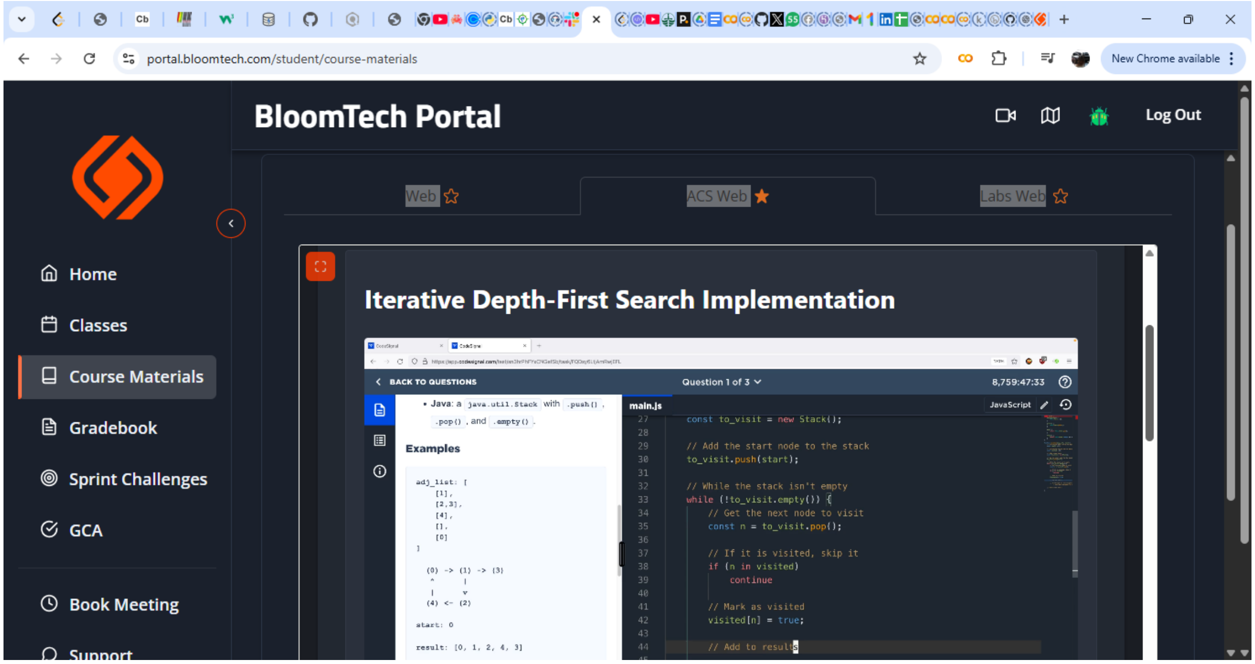
- Java: a `java.util.Stack` with `.push()`, `.pop()`, and `.empty()`.

Examples

```
adj_list: [
    [1],
    [2,3],
    [4],
    [],
    [0]
]

  (0) -> (1) -> (3)
   ^      |
   |      v
  (4) <- (2)

start: 0

result: [0, 1, 2, 4, 3]
```

```
    const result = [];

    // Initialize visited set to empty
    const visited = {};

    // Make a new stack
    const to_visit = new Stack();

    // Add the start node to the stack
    to_visit.push(start);

    // While the stack isn't empty
    while (!to_visit.empty()) {
        // Get the next node to visit
        const n = to_visit.pop();

        // If it is visited
```

Let's see, if n is in visited, we've



portal.bloomtech.com/student/course-materials

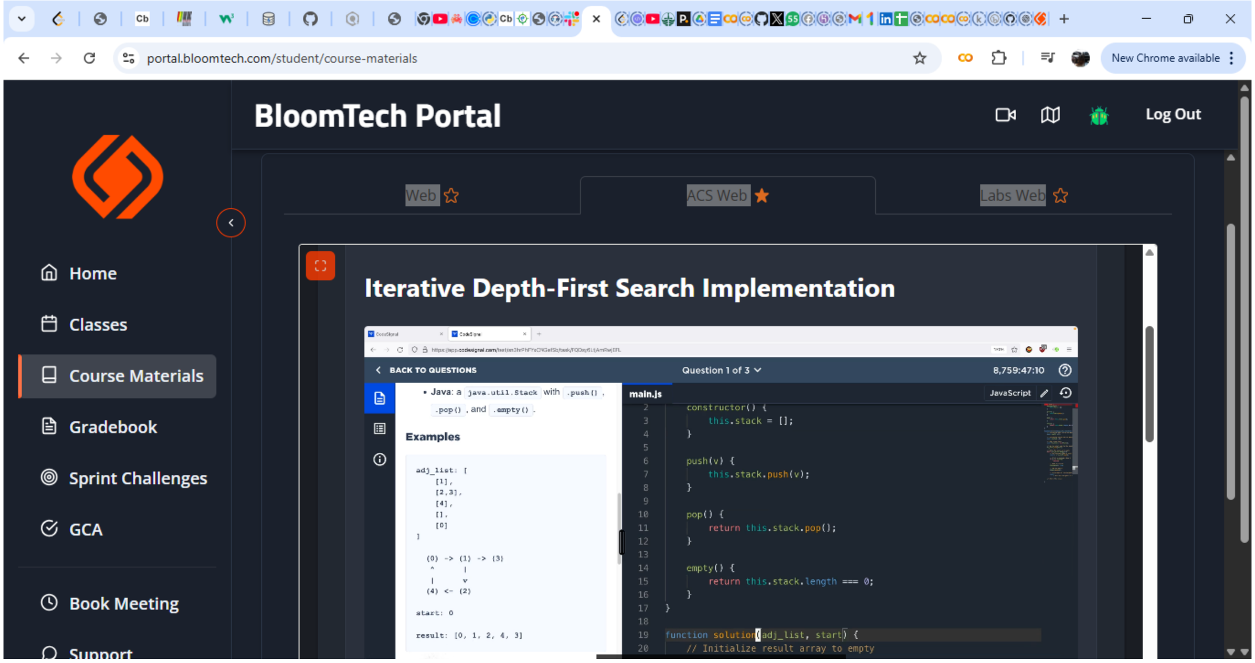
Iterative Depth-First Search Implementation

- Java: a `java.util.Stack` with `.push()`, `.pop()`, and `.empty()`.

Examples

```
adj_list: [
    [1],
    [2,3],
    [4],
    [],
    [0]
]

  (0) -> (1) -> (3)
   ^       |
   |       v
  (4) <- (2)

start: 0

result: [0, 1, 2, 4, 3]
```

```
    const to_visit = new Stack();

    // Add the start node to the stack
    to_visit.push(start);

    // While the stack isn't empty
    while (!to_visit.empty()) {
        // Get the next node to visit
        const n = to_visit.pop();

        // If it is visited, skip it
        if (n in visited)
            continue

        // Mark as visited
        visited[n] = true;

        // Add to results
```



portal.bloomtech.com/student/course-materials

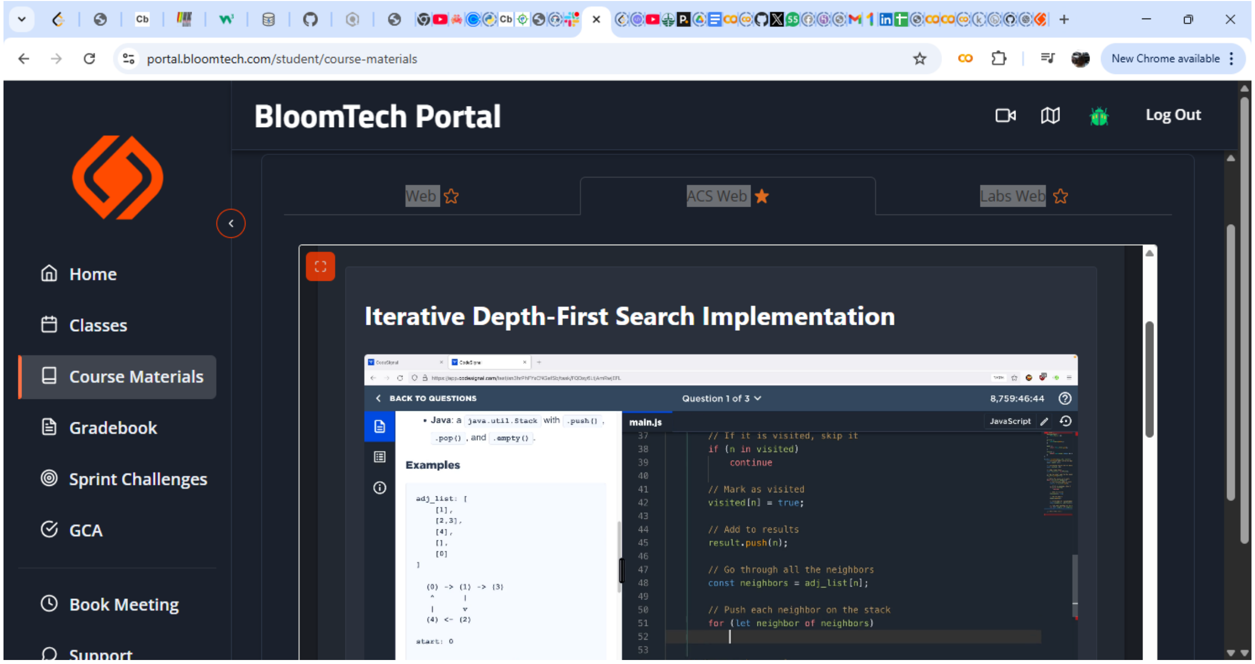
Iterative Depth-First Search Implementation

- Java: a `java.util.Stack` with `.push()`, `.pop()`, and `.empty()`.

Examples

```
adj_list: [
    [1],
    [2,3],
    [4],
    [],
    [0]
]

  (0) -> (1) -> (3)
   ^       |
   |       v
  (4) <- (2)

start: 0

result: [0, 1, 2, 4, 3]
```

```
    constructor() {
        this.stack = [];
    }

    push(v) {
        this.stack.push(v);
    }

    pop() {
        return this.stack.pop();
    }

    empty() {
        return this.stack.length === 0;
    }
}

function solution(adj_list, start) {
    // Initialize result array to empty
```



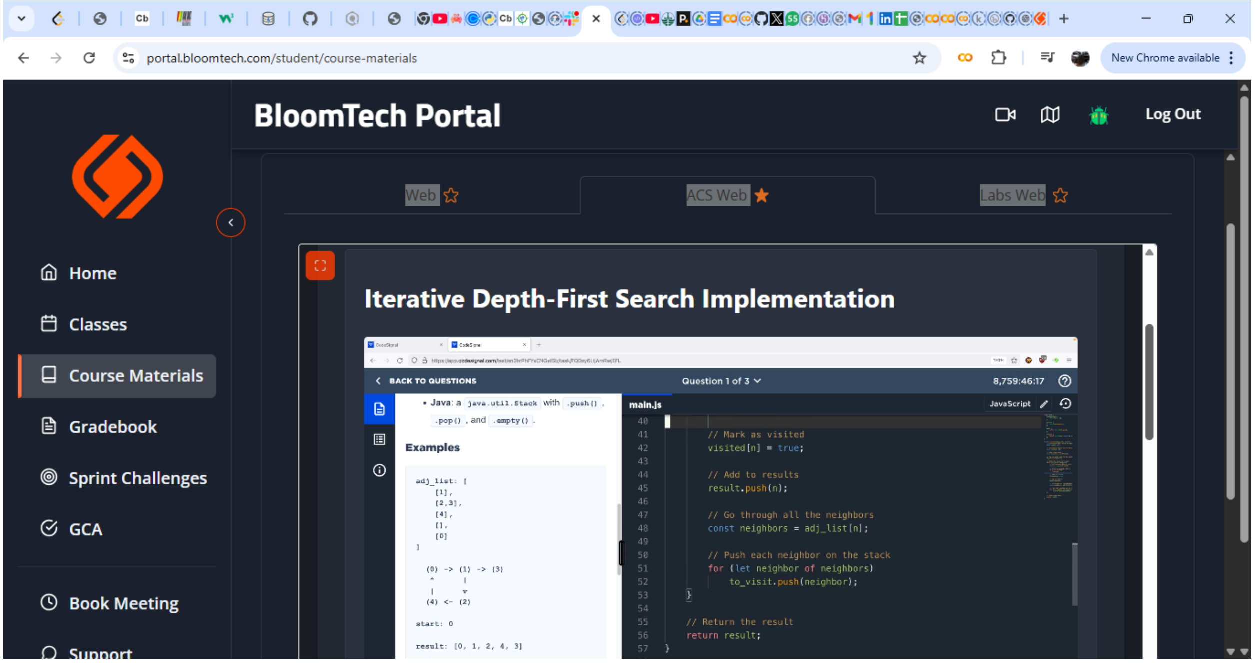
Iterative Depth-First Search Implementation

- Java: a `java.util.Stack` with `.push()`, `.pop()`, and `.empty()`.

Examples

```
adj_list: [
    [1],
    [2,3],
    [4],
    [],
    [0]
]

  (0) -> (1) -> (3)
   ^       |
   |       v
  (4) <- (2)

start: 0
```

```
        // If it is visited, skip it
        if (n in visited)
            continue

        // Mark as visited
        visited[n] = true;

        // Add to results
        result.push(n);

        // Go through all the neighbors
        const neighbors = adj_list[n];

        // Push each neighbor on the stack
        for (let neighbor of neighbors)
            

```



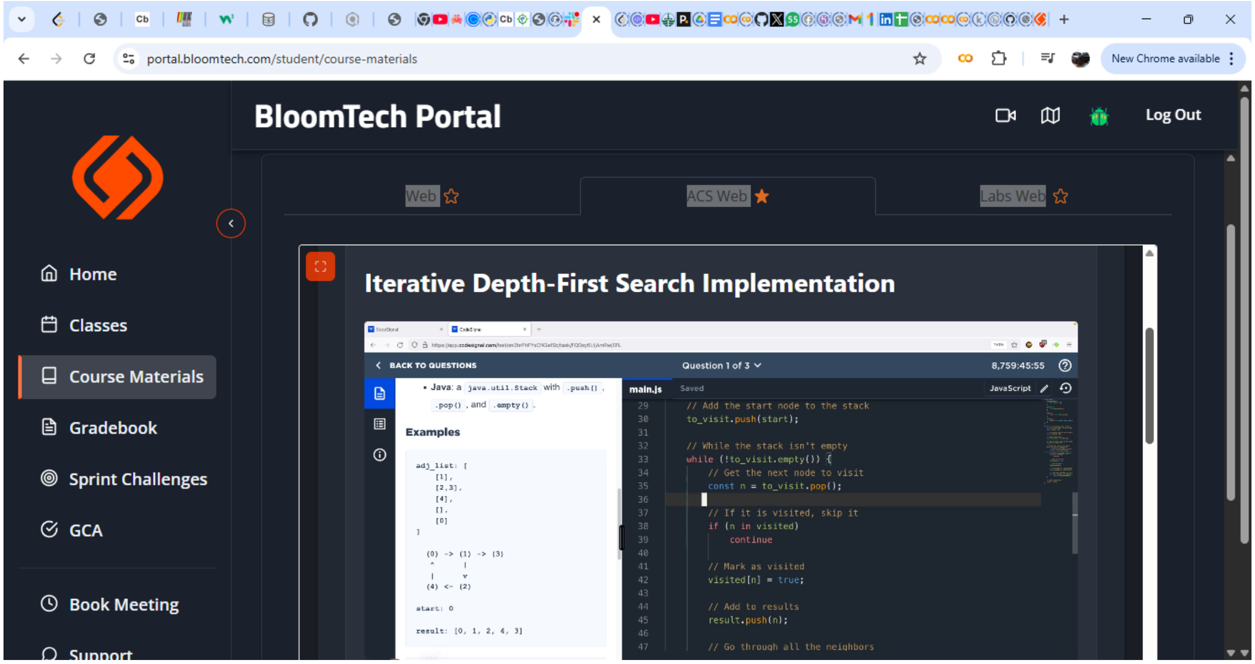
Iterative Depth-First Search Implementation

- Java: a `java.util.Stack` with `.push()`, `.pop()`, and `.empty()`.

Examples

```
adj_list: [
    [1],
    [2,3],
    [4],
    [],
    [0]
]

  (0) -> (1) -> (3)
   ^       |
   |       v
  (4) <- (2)

start: 0

result: [0, 1, 2, 4, 3]
```

```

        // Mark as visited
        visited[n] = true;

        // Add to results
        result.push(n);

        // Go through all the neighbors
        const neighbors = adj_list[n];

        // Push each neighbor on the stack
        for (let neighbor of neighbors)
            to_visit.push(neighbor);
    }

    // Return the result
    return result;
}
```



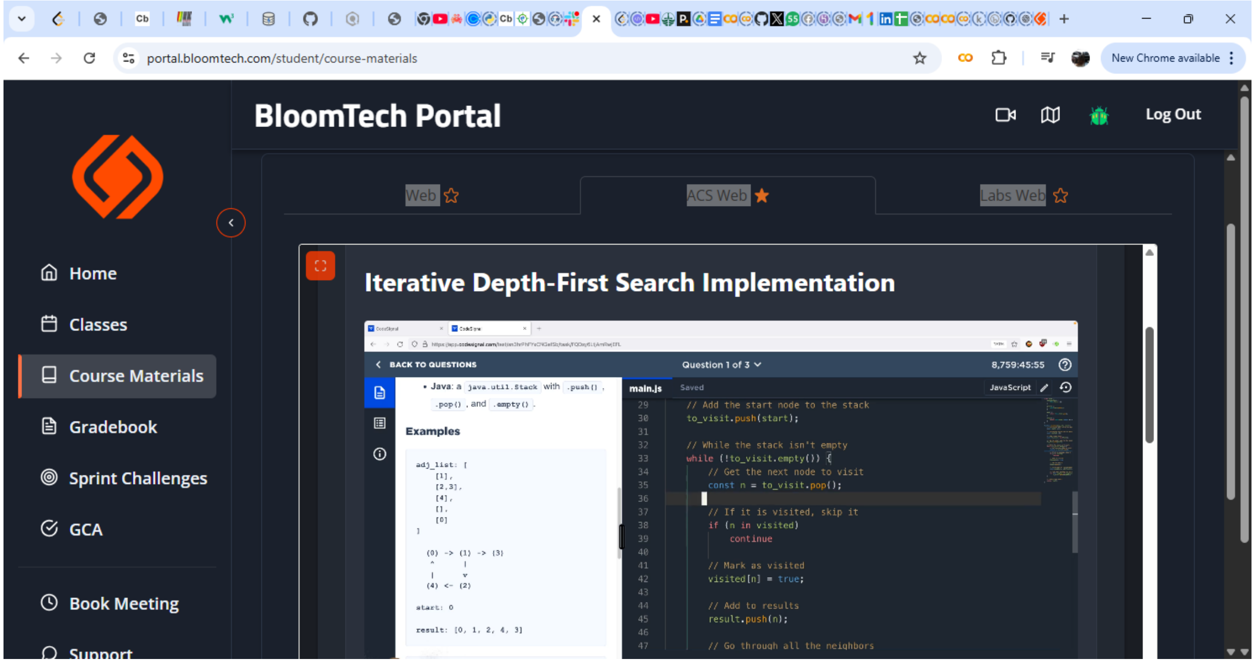
Iterative Depth-First Search Implementation

- Java: a `java.util.Stack` with `.push()`, `.pop()`, and `.empty()`.

Examples

```
adj_list: [
    [1],
    [2,3],
    [4],
    [],
    [0]
]

  (0) -> (1) -> (3)
   ^      |
   |      v
  (4) <- (2)

start: 0

result: [0, 1, 2, 4, 3]
```

```
    // Add the start node to the stack
    to_visit.push(start);

    // While the stack isn't empty
    while (!to_visit.empty()) {
        // Get the next node to visit
        const n = to_visit.pop();

        // If it is visited, skip it
        if (n in visited)
            continue

        // Mark as visited
        visited[n] = true;

        // Add to results
        result.push(n);

        // Go through all the neighbors
```

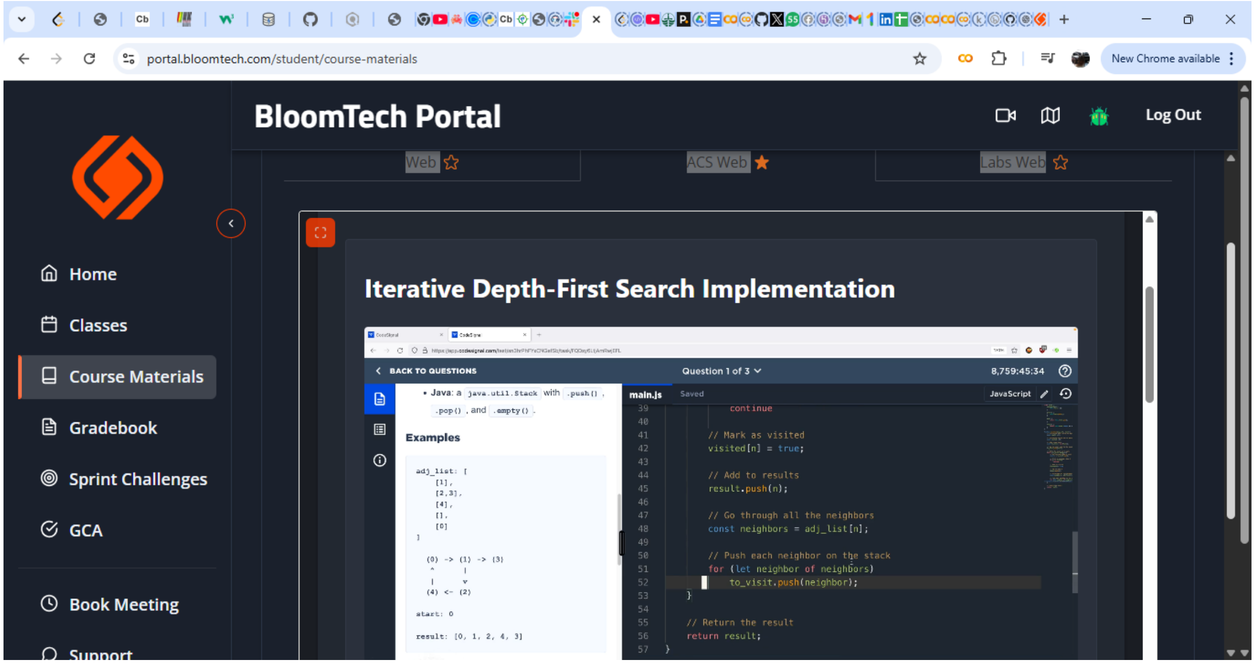

Iterative Depth-First Search Implementation

- Java: a `java.util.Stack` with `.push()`, `.pop()`, and `.empty()`.

Examples

```
adj_list: [
    [1],
    [2,3],
    [4],
    [],
    [0]
]

  (0) -> (1) -> (3)
   ^      |
   |      v
  (4) <- (2)

start: 0

result: [0, 1, 2, 4, 3]
```

```
    // Add the start node to the stack
    to_visit.push(start);

    // While the stack isn't empty
    while (!to_visit.empty()) {
        // Get the next node to visit
        const n = to_visit.pop();

        // If it is visited, skip it
        if (n in visited)
            continue

        // Mark as visited
        visited[n] = true;

        // Add to results
        result.push(n);

        // Go through all the neighbors
```



Iterative Depth-First Search Implementation

- Java: a `java.util.Stack` with `.push()`, `.pop()`, and `.empty()`.

Examples

```
adj_list: [
    [1],
    [2,3],
    [4],
    [],
    [0]
]

  (0) -> (1) -> (3)
   ^       |
   |       v
  (4) <- (2)

start: 0

result: [0, 1, 2, 4, 3]
```

```
            continue

        // Mark as visited
        visited[n] = true;

        // Add to results
        result.push(n);

        // Go through all the neighbors
        const neighbors = adj_list[n];

        // Push each neighbor on the stack
        for (let neighbor of neighbors)
            to_visit.push(neighbor);
    }

    // Return the result
    return result;
}
```