Further List Operations

Let's explore more features of the list data structure!

WE'LL COVER THE FOLLOWING

Appending a List

A list can be appended to another list using the spread operator. Here's how it works:

```
let myList = [5, 6, 7, 8];

Js.log([1, 2, 3, 4, ...myList]);
```

> **Console**: `[1,[2,[3,[4,[5,[6,[7,[8,0]]]]]]]]`

However, the code below would not create the correct structure:

```
let list1 = [1, 2, 3, 4];
let list2 = [5, 6, 7, 8];

Js.log([list1, ...[list2]]);
```

> **Output**: `[[1,[2,[3,[4,0]]]],[[5,[6,[7,[8,0]]]],0]]`

Instead of one merged list, we get a list of lists containing `list1` and `list2`.

We could define a recursive function and use the list's variant property to join two lists:

```
let list1 = [1, 2, 3, 4];
let list2 = [5, 6, 7, 8];

let rec join = (list1, list2) => {
  switch(list1) {
    | [] => list2
    | [head, ...tail] => [head, ...join(tail, list2)]
  };
};

Js.log(join(list1, list2));
```

> **Console**: `[1,[2,[3,[4,[5,[6,[7,[8,0]]]]]]]]`

Alternatively, we can use the `@` operator to concatenate two lists as well:

```
let list1 = [1, 2, 3, 4];
let list2 = [5, 6, 7, 8];

Js.log(list1 @ list2);
```

> **Console**: `[1,[2,[3,[4,[5,[6,[7,[8,0]]]]]]]]`

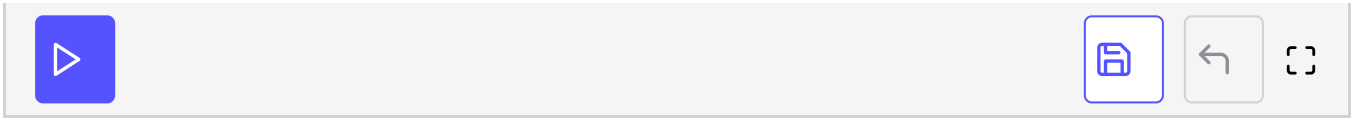
Accessing the Head

We can access the first element of the list using pattern matching:

```
let myList = ["Reason", "JavaScript", "OCaml"];

let getHead = (myList) => {
  switch(myList){
    | [] => ""
    | [head, ...tail] => head
  }
};

Js.log(getHead(myList));
```

Reason also has a predefined function, `List.hd()`, for the process above:

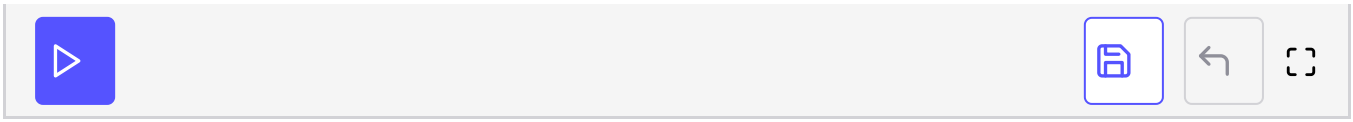
```
let myList = ["Reason", "JavaScript", "OCaml"];

Js.log(List.hd(myList));
```



Accessing the N-th Element

The last element of a list can be accessed using `List.nth()`. The second argument of this function is the *nth* index at which we want to find the value:

```
let myList = ["Reason", "JavaScript", "OCaml"];

Js.log(List.nth(myList, 2));
```



Simple Iteration

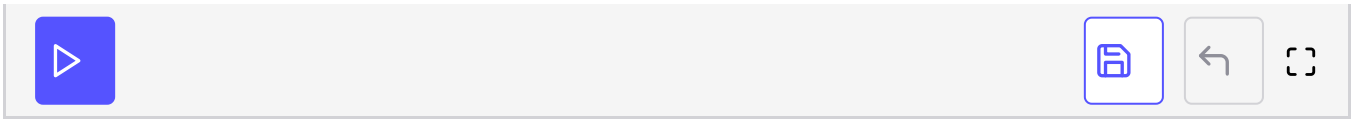
We can use the `List.iter()` function to iterate over the list. Below, we can see the template of this function:

```
iter(f, list('a))
```

`f` is a function which will be applied to all the elements being iterated.

Here's an example:

```
let myList = ["Reason", "JavaScript", "OCaml"];

let f = (str: string) => Js.log(str);

List.iter(f, myList);
```



To see all the functions of the list data structure, check out the official

To see all the functions of the list data structure, check out the official documentation here.

In the next lesson, we'll test our theoretical understanding of variants and lists with a quiz.