Stack

What is a Stack?

Lists are a natural form of organization for data. We have already seen how to use the List class to organize data into a list. When the order of the data being stored doesn't matter, or when you don't have to search the data stored, lists work wonderfully. For other applications, however, plain lists are too simple and we need a more complex, list- like data structure.

A list-like structure that can be used to solve many problems in computing is the stack. Stacks are efficient data structures because data can be added or removed only from the top of a stack, making these procedures fast and easy to implement. Stacks are used extensively in programming language implementations for everything from expression evaluation to handling function calls.

Stack Operations

A stack is a list of elements that are accessible only from one end of the list, which is called the top. One common, real-world example of a stack is the stack of trays at a cafeteria. Trays are always removed from the top, and when trays are put back on the stack after being washed, they are placed on the top of the stack. The stack is known as a last-in, first-out (LIFO) data structure.

Because of the last-in, first-out nature of the stack, any element that is not currently at the top of the stack cannot be accessed. To get to an element at the bottom of the stack, you have to dispose of all the elements above it first.

The two primary operations of a stack are adding elements to a stack and taking elements off a stack. Elements are added to a stack using the push operation. Elements are taken off a stack using the pop operation. These operations are illustrated in the below figure..

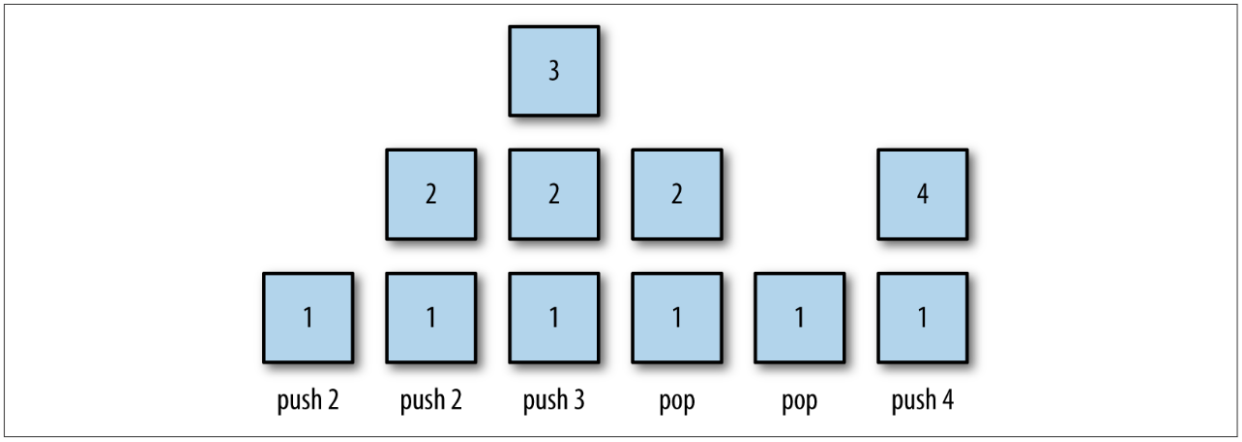

Figure shows pushing and popping elements of a stack

Another common operation on a stack is viewing the element at the top of a stack. The pop operation visits the top element of a stack, but it permanently removes the element from a stack. The peek operation returns the value stored at the top of a stack without removing it from the stack.

To keep track of where the top element is, as well as keeping track of where to add a new element, we use a top variable that is incremented when we push new elements onto the stack and is decremented when we pop elements off the stack.

While pushing, popping, and peeking are the primary operations associated with a stack, there are other operations we need to perform and properties we need to examine. The clear operation removes all the elements from a stack. The length property holds the number of elements contained in a stack. We also define an empty property to let us know if a stack has no elements in it, though we can use the length property for this as well.

A Stack Implementation

To build a stack, we first need to decide on the underlying data structure we will use to store the stack elements. We will use an array in our implementation.

We begin our stack implementation by defining the constructor function for a Stack class:

```
function Stack() {
this.dataStore = [];
this.top = 0;
this.push = push;
this.pop = pop;
this.peek = peek;
}
```

The array that stores the stack elements is named dataStore. The constructor sets it to an empty array. The top variable keeps track of the top of the stack and is initially set to 0 by the constructor, indicating that the 0 position of the array is the top of the stack, at least until an element is pushed onto the stack.

The first function to implement is the push() function. When we push a new element onto a stack, we have to store it in the top position and increment the top variable so that the new top is the next empty position in the array. Here is the code:

```
function push(element) { this.dataStore[this.top++] = element;}
```

Pay particular attention to the placement of the increment operator after the call to this.top. Placing the operator there ensures that the current value of top is used to place the new element at the top of the stack before top is incremented.

The pop() function does the reverse of the push() function—it returns the element in the top position of the stack and then decrements the top variable:

```
function pop() {
return this.dataStore[--this.top];
}
```

The peek() function returns the top element of the stack by accessing the element at the top-1 position of the array:

```
function peek() {
return this.dataStore[this.top-1];
}
```

If you call the peek() function on an empty stack, you get undefined as the result. That's because there is no value stored at the top position of the stack since it is empty.

There will be situations when you need to know how many elements are stored in a stack. The length() function returns this value by returning the value of top:

```
function length() { return this.top;}
```

Finally, we can clear a stack by simply setting the top variable back to 0:

```
function clear() {
this.top = 0;
}
```

The Stack class

```
function Stack() {
this.dataStore = [];
this.top = 0;
this.push = push;
this.pop = pop;
this.peek = peek;
this.clear = clear;
this.length = length;

function push(element) {
this.dataStore[this.top++] = element;
}

function peek() {
return this.dataStore[this.top-1];
}

function pop() {
return this.dataStore[--this.top];
}

function clear() {
this.top = 0;
}

function length() {
return this.top;
}
```

Example of Stack class implementation

```
var s = new Stack();
s.push("David");
s.push("Raymond");
s.push("Bryan");
print("length: " + s.length());
print(s.peek());
var popped = s.pop();
print("The popped element is: " + popped); print(s.peek());
s.push("Cynthia");
print(s.peek());
s.clear();
print("length: " + s.length()); print(s.peek());
s.push("Clayton");
print(s.peek());
```

Output:

length: 3

Bryan

The popped element is: Bryan

Raymond

Cynthia

length: 0

undefined

Clayton

The next-to-last value, undefined, is returned because once a stack is cleared, there is no value in the top position and when we peek at the top of the stack, undefined is returned.

Note: Stacks support peek, insertion, and deletion in O(1). A stack follows a LIFO structure that is Last In First Out.