Stack (Implementation)

In this lesson, we are going to look at how Stacks are implemented in Python and how the main Stack functions actually work

We'll cover the following

- Introduction
- Implementation
- Complexities of Stack Operations

Introduction

Most programming languages come with the Stack data structure built in. In Python, you can use the pre-built Stack class by importing them into your program. However, implementing a stack from scratch will allow you to truly master the ins and outs of the data structure.

Implementation

Stacks can be implemented using Lists or Linked Lists. Each implementation has its own advantages and disadvantages. Here, however, we will show an implementation of stacks using lists.

As mentioned in the previous lesson, a typical Stack must contain the following functions:

- `push(element)`
- `pop()`
- `is_empty()`
- `top()`
- `size()`

We will take a close look at these functions individually, but, before we do, let's construct a Stack class and create an object. This class will consist of the member functions given above and a list that will hold all the elements of the stack.

Stack.py

```python
class MyStack:
    def __init__(self):
        self.stack_list = []


stack = MyStack()

```

Now, before adding the `push(element)` and `pop()` functions into this code, let’s implement the following functions:

- `is_empty()`
- `top()`
- `size()`

Examine the following code.

Stack.py

```python
class MyStack:
    def __init__(self):
        self.stack_list = []

    def is_empty(self):
        return self.size() == 0

    def top(self):
        if self.is_empty():
            return None
        return self.stack_list[-1]

    def size(self):
        return len(self.stack_list)


stack = MyStack()
print("is_empty(): " + str(stack.is_empty()))
print("top(): " + str(stack.top()))
print("size(): " + str(stack.size()))

```

The `is_empty()` function returns the boolean you get from comparing the length of the array to 0. If the length of the array is 0, then the function will return `True`. Otherwise, it will return `False`.

If the list is not empty, the `top()` function returns the last element in the list, which we are considering to be the *top* of the stack!

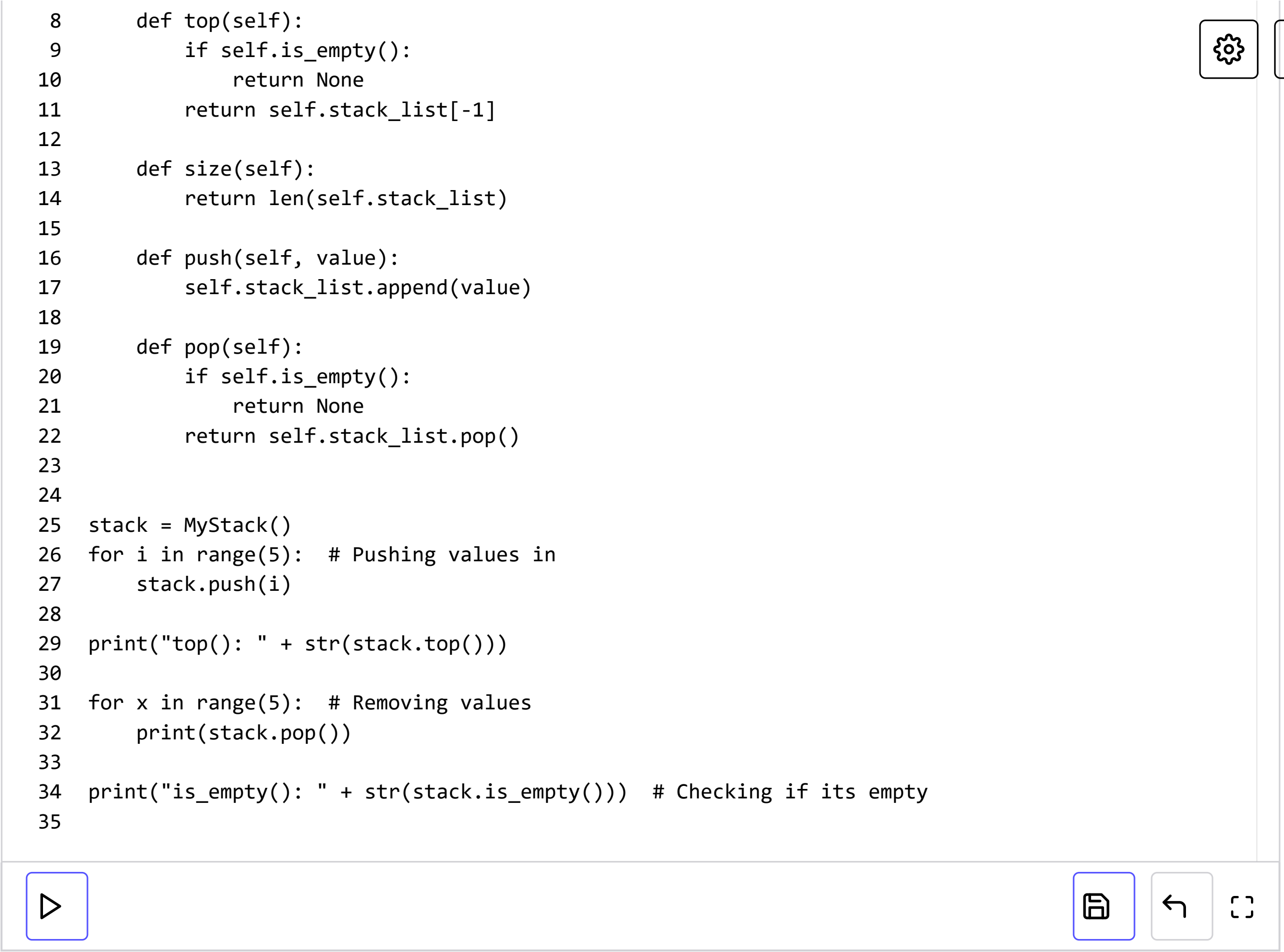
Now, study the implementation of the `push(element)` and `pop()` functions.

Stack.py

```python
class MyStack:
    def __init__(self):
        self.stack_list = []

    def is_empty(self):
        return len(self.stack_list) == 0

```



```
    def top(self):
        if self.is_empty():
            return None
        return self.stack_list[-1]

    def size(self):
        return len(self.stack_list)

    def push(self, value):
        self.stack_list.append(value)

    def pop(self):
        if self.is_empty():
            return None
        return self.stack_list.pop()


stack = MyStack()
for i in range(5):  # Pushing values in
    stack.push(i)

print("top(): " + str(stack.top()))

for x in range(5):  # Removing values
    print(stack.pop())

print("is_empty(): " + str(stack.is_empty()))  # Checking if its empty

```

If you look at the output of the code, you can see that the elements popped out of the stack in the exact reverse order that they were pushed in. That means our Stack works perfectly. Congratulations, you have successfully implemented a Stack using a Python List!

Complexities of Stack Operations

Let’s look at the time complexity of each stack operation.

Operation	Time Complexity
is_empty	O(1)
top	O(1)
size	O(1)
push	O(1)
pop	O(1)

The next data structure that we are going to look at is a **Queue**.

← Back

Next →

☑ Mark as Completed

Report an Issue

Ask a Question
(https://discuss.educative.io/tag/stack-implementation__introduction-to-stackqueues__data-structures-for-coding-interviews-in-python)