# 3.5. Implementing a Stack in Python

Now that we have clearly defined the stack as an abstract data type we will turn our attention to using Python to implement the stack. Recall that when we give an abstract data type a physical implementation we refer to the implementation as a data structure.

As we described in Chapter 1, in Python, as in any object-oriented programming language, the implementation of choice for an abstract data type such as a stack is the creation of a new class. The stack operations are implemented as methods. Further, to implement a stack, which is a collection of elements, it makes sense to utilize the power and simplicity of the primitive collections provided by Python. We will use a list.

Recall that the list class in Python provides an ordered collection mechanism and a set of methods. For example, if we have the list [2,5,3,6,7,4], we need only to decide which end of the list will be considered the top of the stack and which will be the base. Once that decision is made, the operations can be implemented using the list methods such as `append` and `pop` .

The following stack implementation (ActiveCode 1) assumes that the end of the list will hold the top element of the stack. As the stack grows (as `push` operations occur), new items will be added on the end of the list. `pop` operations will manipulate that same end.

Run Load History

```
class Stack:
     def __init__(self):
         self.items = []

     def isEmpty(self):
         return self.items == []

     def push(self, item):
         self.items.append(item)

     def pop(self):
         return self.items.pop()

     def peek(self):
         return self.items[len(self.items)-1]

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

```

**ıckAbstractDataType.html)**

ActiveCode: 1 Implementing a Stack class using Python lists (stack_1ac)

Remember that nothing happens when we click the `run` button other than the definition of the class. We must create a `Stack` object and then use it. ActiveCode 2 shows the `Stack` class in action as we perform the sequence of operations from Table 1 (TheStackAbstractDataType.html#tbl-stackops). Notice that the definition of the `Stack` class is imported from the `pythonds` module.

**Note**

The `pythonds` module contains implementations of all data structures discussed in this book. It is structured according to the sections: basic, trees, and graphs. The module can be downloaded from pythonworks.org (http://www.pythonworks.org/pythonds).

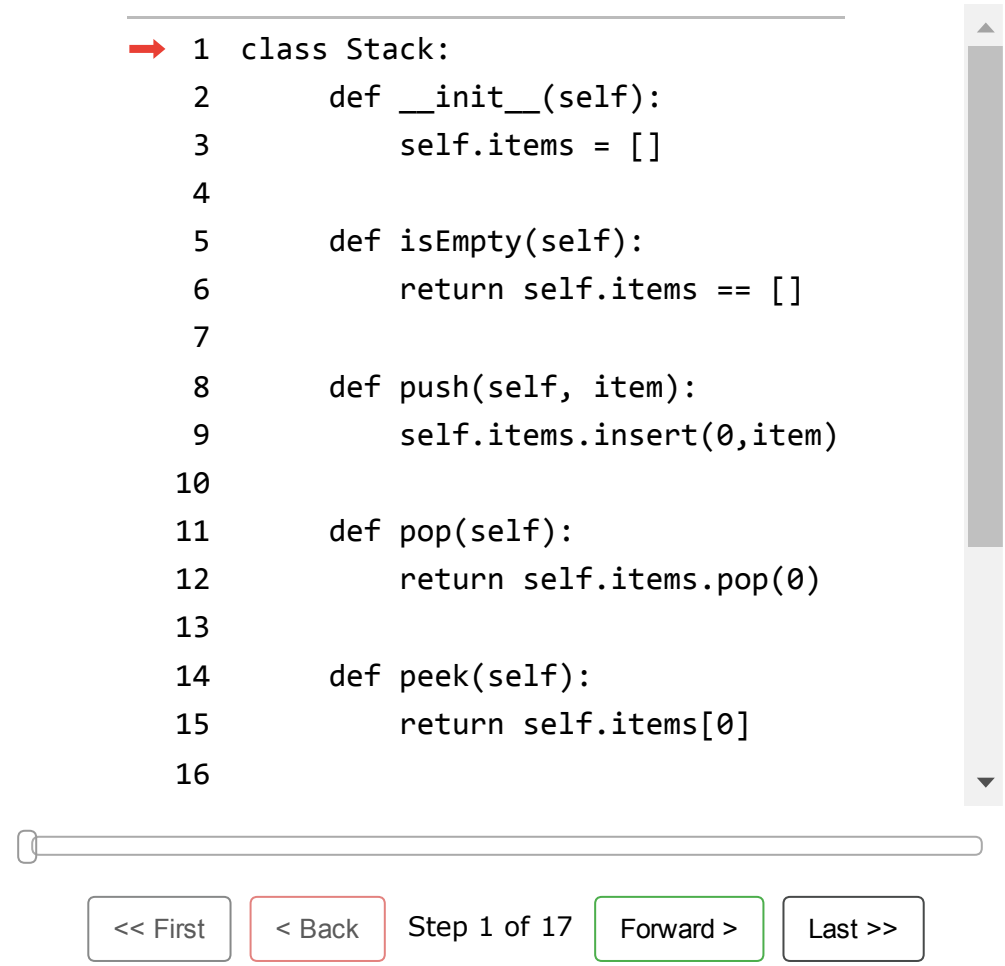
Run Load History

```
1 from pythonds.basic.stack import Stack
2
3 s=Stack()
4
5 print(s.isEmpty())
6 s.push(4)
7 s.push('dog')
8 print(s.peek())
9 s.push(True)
10 print(s.size())
11 print(s.isEmpty())
12 s.push(8.4)
13 print(s.pop())
14 print(s.pop())
15 print(s.size())
16
```

ActiveCode: 2 (stack_ex_1)

It is important to note that we could have chosen to implement the stack using a list where the top is at the beginning instead of at the end. In this case, the previous `pop` and `append` methods would no longer work and we would have to index position 0 (the first item in the list) explicitly using `pop` and `insert` . The implementation is shown in CodeLens 1.

**ıckAbstractDataType.html)**




**CodeLens: 1 Alternative Implementation of the Stack class (stack_cl_1)**

This ability to change the physical implementation of an abstract data type while maintaining the logical characteristics is an example of abstraction at work. However, even though the stack will work either way, if we consider the performance of the two implementations, there is definitely a difference. Recall that the `append` and `pop()` operations were both O(1). This means that the first implementation will perform push and pop in constant time no matter how many items are on the stack. The performance of the second implementation suffers in that the `insert(0)` and `pop(0)` operations will both require O(n) for a stack of size n. Clearly, even though the implementations are logically equivalent, they would have very different timings when performing benchmark testing.

ıckAbstractDataType.html)

**Self Check**

Q-7: Given the following sequence of stack operations, what is the top item on the stack when the sequence is complete?

```
m = Stack()
m.push('x')
m.push('y')
m.pop()
m.push('z')
m.peek()
```

- (A) 'x'
- (B) 'y'
- (C) 'z'
- (D) The stack is empty

Check Me | Compare me

Q-8: Given the following sequence of stack operations, what is the top item on the stack when the sequence is complete?

```
m = Stack()
m.push('x')
m.push('y')
m.push('z')
while not m.isEmpty():
   m.pop()
   m.pop()
```

- (A) 'x'
- (B) the stack is empty
- (C) an error will occur
- (D) 'z'

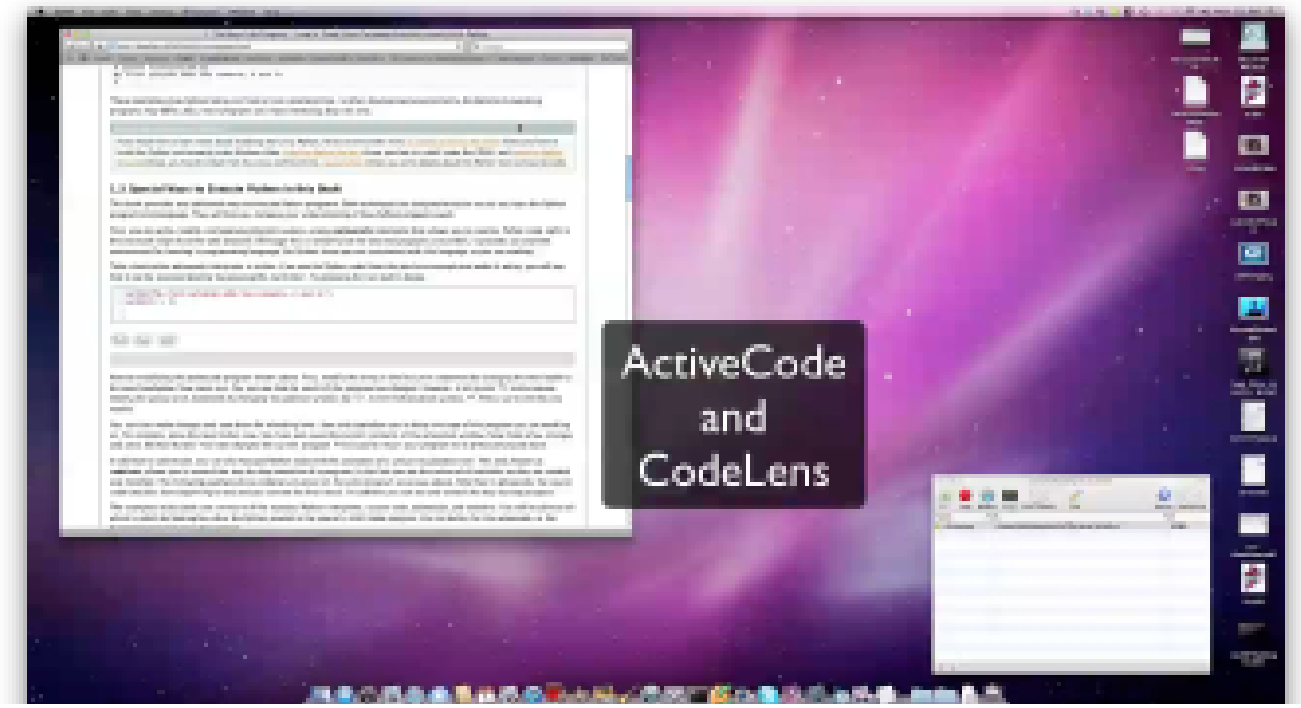
Check Me | Compare me

**ıckAbstractDataType.html)**

Write a function revstring(mystr) that uses a stack to reverse the characters in a string.

Run | Show Feedback | Show Code | Code Coach

ActiveCode: 3 (stack_stringrev)




0:00

❮ (TheStackAbstractDataType.html) ❯ (SimpleBalancedParentheses.html)

user not logged in

**ıckAbstractDataType.html)**