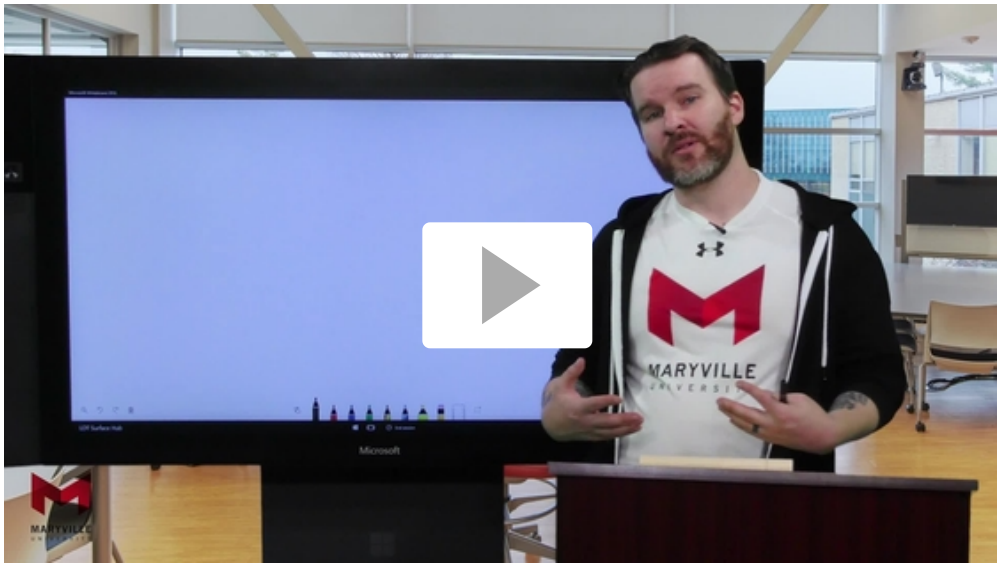
Stacks

Let's begin by watching the video "Stacks" (3:56).



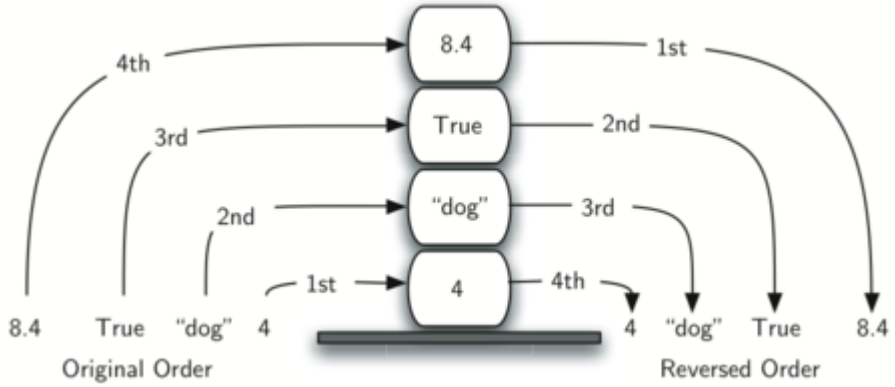
A **stack** (sometimes called a “push-down stack”) is an ordered collection of items where the addition of new items and the removal of existing items always takes place at the same end. This end is commonly referred to as the “top.” The end opposite the top is known as the “base.”

The base of the stack is significant since items stored in the stack that are closer to the base represent those that have been in the stack the longest. The most recently added item is the one that is in position to be removed first. This ordering principle is sometimes called **LIFO**, **last-in first-out**. It provides an ordering based on length of time in the collection. Newer items are near the top, while older items are near the base.

Many examples of stacks occur in everyday situations. Almost any cafeteria has a stack of trays or plates where you take the one at the top, uncovering a new tray or plate for the next customer in line. Imagine a stack of books on a desk. The only book whose cover is visible is the one on top. To access others in the stack, we need to remove the ones that are sitting on top of them.

The Reversal Property of Stacks

One of the most useful ideas related to stacks comes from the simple observation of items as they are added and then removed. Assume you start out with a clean desktop. Now place books one at a time on top of each other. You are constructing a stack. Consider what happens when you begin removing books. The order that they are removed is exactly the reverse of the order that they were placed. Stacks are fundamentally important, as they can be used to reverse the order of items. The order of insertion is the reverse of the order of removal. The picture below shows the Python data object stack as it was created and then again as items are removed. Note the order of the objects.



Considering this reversal property, you can perhaps think of examples of stacks that occur as you use your computer. For example, every web browser has a Back button. As you navigate from web page to web page, those pages are placed on a stack (actually it is the URLs that are going on the stack). The current page that you are viewing is on the top and the first page you looked at is at the base. If you click on the Back button, you begin to move in reverse order through the pages.

The Stack Abstract Data Type

The stack abstract data type is defined by the following structure and operations. A stack is structured, as described above, as an ordered collection of items where items are added to and removed from the end called the "top." Stacks are ordered LIFO. The stack operations are given below.

- `Stack()` creates a new stack that is empty. It needs no parameters and returns an empty stack.

- `push(item)` adds a new item to the top of the stack. It needs the item and returns nothing.
- `pop()` removes the top item from the stack. It needs no parameters and returns the item. The stack is modified.
- `peek()` returns the top item from the stack but does not remove it. It needs no parameters. The stack is not modified.
- `isEmpty()` tests to see whether the stack is empty. It needs no parameters and returns a boolean value.
- `size()` returns the number of items on the stack. It needs no parameters and returns an integer.

Implementing a Stack in Python

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`.

Here is an example of implementing a stack:

```
from pythonds.basic.stack import Stack

s=Stack()

print(s.isEmpty())
s.push(4)
s.push('dog')
print(s.peek())
s.push(True)
print(s.size())
print(s.isEmpty())
s.push(8.4)
print(s.pop())
print(s.pop())
print(s.size())
```

Source: Problem Solving and Algorithms in Python (http://interactivepython.org/runestone/static/pythonds/index.html#) from Bradley Miller on www.interactivepython.org (http://interactivepython.org/runestone/static/pythonds/index.html#).