Data Structures and Abstract Data Types

- Abstract Data Types
  - Stack
  - Dictionary
- Data structures
  - Array
  - Linked List
  - Tree
- Interface vs. Implementation

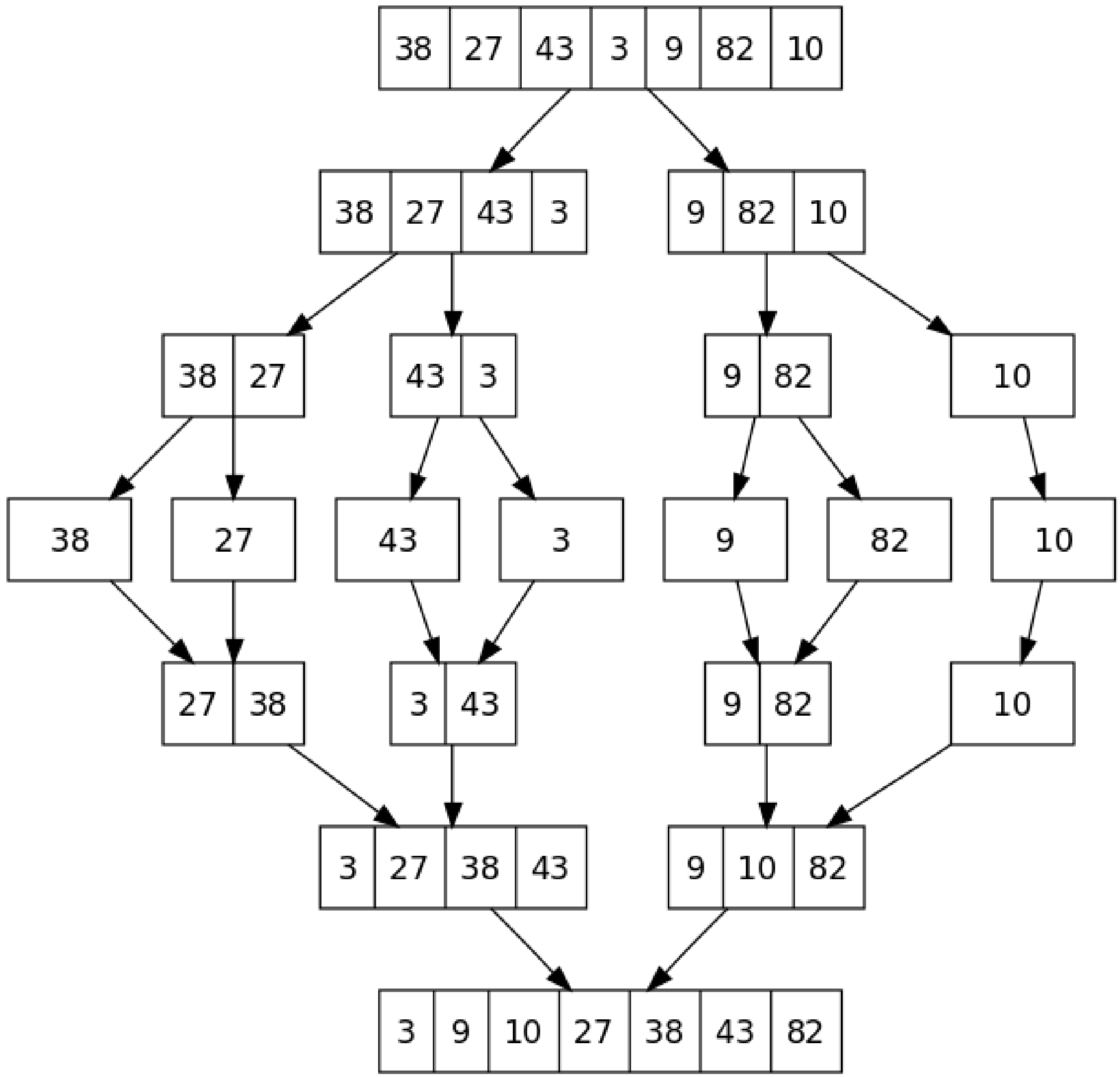
Abstract vs. Concrete

- In any type of programming, we can look at the code on at least two levels
  - Abstract
  - Concrete
- "Abstract" will have to do with the general logic of the code – i.e., the operations/steps that we perform on program data
- "Concrete", then, will concern specific details concerning the particular programming language and its constructs

Abstract vs. Concrete

- Another way to think of this could be the difference between pseudocode and actual code
- Abstract program logic can take concrete form in many different programming languages
- For example, a merge sort generally entails the following logic:
  - Divide the sequence in half
  - Perform a merge sort on each half
  - Merge those two sorted halves back into <u>*one*</u> sorted sequence

Merge sort: Visual



Source: `http://howtodoinjava.com/wp-content/uploads/2015/10/Merge_sort_algorithm.png`

Abstract vs. Concrete

- Depending on the specific programming language, that logic will take on a certain concrete form
- In **sort_code.py**, you can see a Python-based merge sort function
- One notable feature is that Python allows you to define functions (such as `merge`) inside of other functions.
- In contrast, here is a Java-based merge sort...

mergeSort function in Java

```java
public static int[] mergeSort(int [] list) {
    if (list.length <= 1) {
        return list;
    }

    // Split the array in half
    int[] first = new int[list.length / 2];
    int[] second = new int[list.length - first.length];
    System.arraycopy(list, 0, first, 0, first.length);
    System.arraycopy(list, first.length, second, 0, second.length);

    // Sort each half
    mergeSort(first);
    mergeSort(second);

    // Merge the halves together, overwriting the original array
    merge(first, second, list);
    return list;
}
```

Source: `http://javahungry.blogspot.com/2013/06/java-sorting-program-code-merge-sort.html`

merge function in Java

```java
private static void merge(int[] first, int[] second, int [] result) {
    // Merge both halves into the result array
    // Next element to consider in the first array
    int iFirst = 0;
    // Next element to consider in the second array
    int iSecond = 0;

    // Next open position in the result
    int j = 0;
    // As long as neither iFirst nor iSecond is past the end, move the
    // smaller element into the result.
    while (iFirst < first.length && iSecond < second.length) {
        if (first[iFirst] < second[iSecond]) {
            result[j] = first[iFirst];
            iFirst++;
            } else {
            result[j] = second[iSecond];
            iSecond++;
        }
        j++;
    }
    // copy what's left
    System.arraycopy(first, iFirst, result, j, first.length - iFirst);
    System.arraycopy(second, iSecond, result, j, second.length - iSecond);
}
```

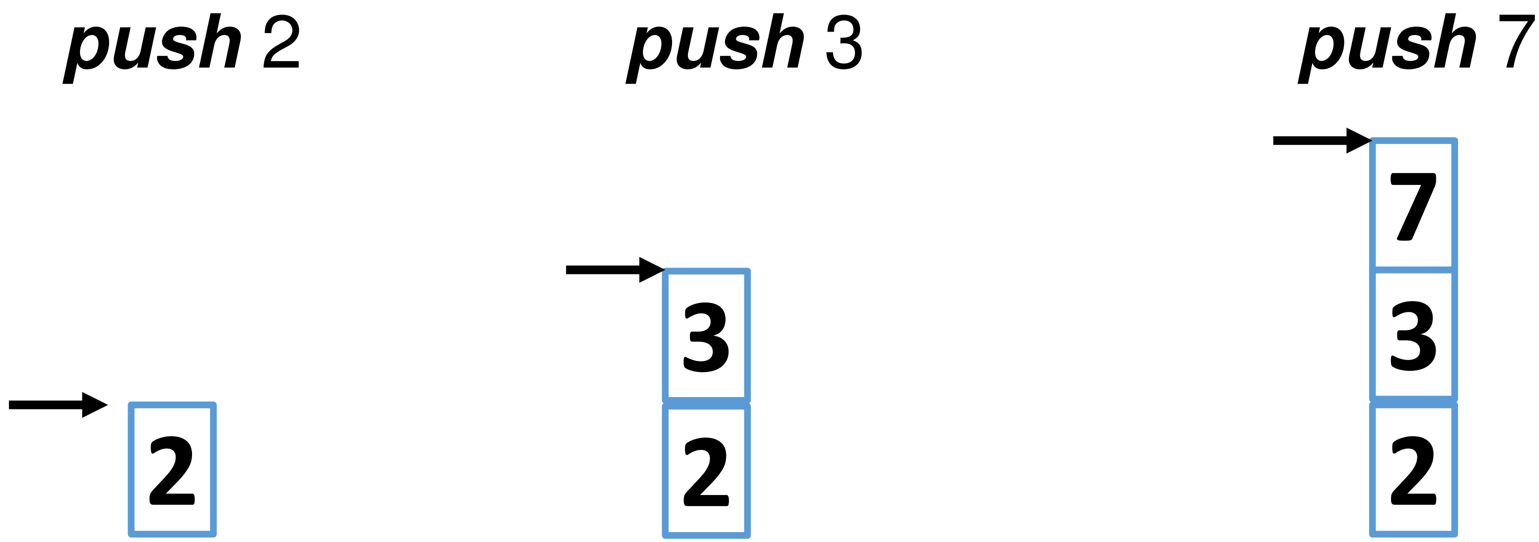
Source: `http://javahungry.blogspot.com/2013/06/java-sorting-program-code-merge-sort.html`

Abstract Data Types

- Furthermore, as with many algorithms, there are variations on merge sort, such that some are more or less efficient – in terms of time or space – than others
- We can also think of certain data types as being abstract. This means that the type has a unique logic that can be defined in general terms.
- We will look at two:
  - Stacks
  - Dictionaries

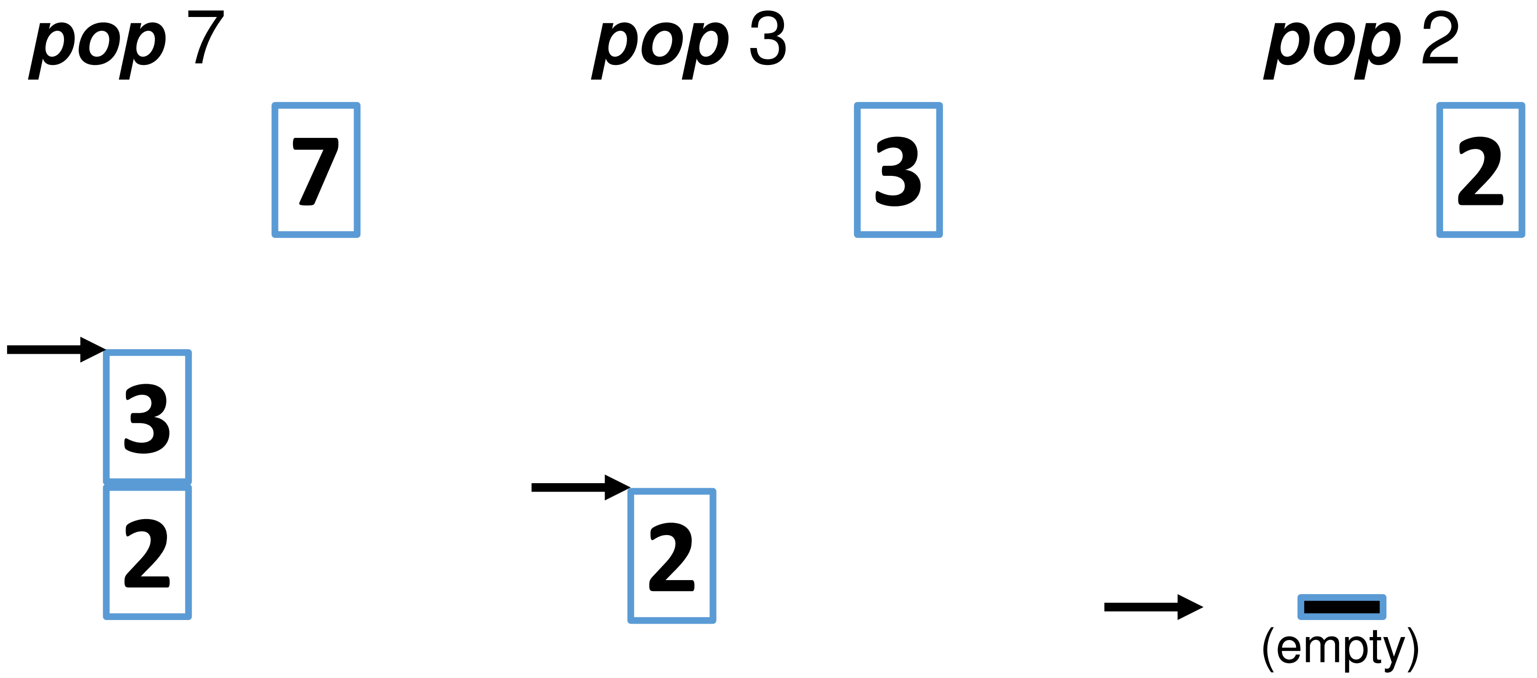
Stacks

- A **stack** is a way of organizing data, defined by the logic of "last in, first out" or LIFO
- This means that the only element in a stack that you can access is the last one you added, or "pushed". For example...



Stacks

- You can think of it as similar to placing items onto a stack, such as eating trays
- When you remove something, we say that you "pop" it from the stack. For example...



Dictionaries

- You are already familiar with the logic of a **dictionary**, by this point.
- Basically, you add data entry in the form of key-value pairs, and you can later access or modify those entries by *key*
- For example...

add **"name", "Joe"**

"name"	"Joe"

add **"age", 22**

"name"	"Joe"
"age"	22

Abstract Data Types

- These and others represent ways of conceptualizing data relationships on an abstract, logical level.
- Many programming languages have stacks, dictionaries, queues, lists, and other such forms
- These abstract data types are defined by their operations
  - Stacks: Push and pop
  - Dictionaries: Adding key-value entries and fetching values by key

Concrete Data Structures

- However, this abstract logic can be achieved in a number of ways, which will depend upon various *concrete* data structures available in a particular programming language.
- These are the specific language- and machine-dependent modes of storing and accessing data
- We will look at three:
  - Array
  - Linked List
  - Tree

Arrays

- An **array** is characterized by a sequence data elements accessible via an index, or position number.
- In Python, this has been in the form of tuples and lists – the main difference being that tuples are immutable while lists are mutable.

```
my_list = [ "hello", "goodbye", "yes", "no" ]
print (my_list[1])
```

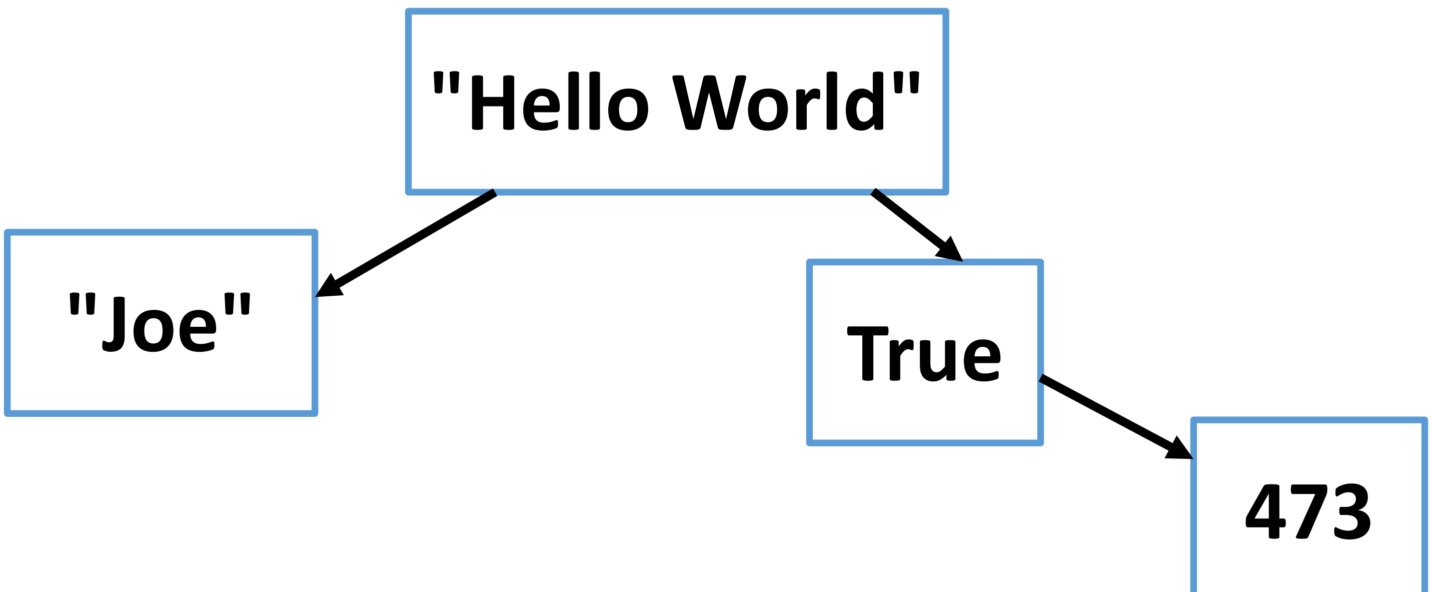
- You will see similar concrete data structure in many programming languages

Linked Lists

- A **linked list** is made up of nodes – where each node, starting with the head, points to the next node.
- The end node, of course, does not point to any next node.
- Each node is a container for a value

Trees

- A **<u>tree</u>** is also made up of nodes – where each node, starting with the <u>root</u>, points to zero or more <u>child nodes</u>
- Nodes that have no child nodes are called <u>leaves</u>.
- Each node is a container for a value



Interface vs. Implementation

- The different abstract data types will have certain operations that you can perform on them:
  - Stack: Push and Pop
  - Dictionary: Add Entry, Fetch Value by Key
- These operations, then, can be considered the interface for their respective types
- The implementation, then, will consist of the underlying concrete data structure, as well as the code to implement the abstract operations.