CSCI 132:
Basic Data Structures and Algorithms

ArrayLists

Reese Pearsall

Spring 2023

Announcements

Lab 4 due **TUESDAY** at 11:59 PM

- Interfaces.

Gideon (8 AM, 10 AM) is out sick this week

```
Roses are Red,
Violets are Blue.

Unexpected ‘{’ on line 32.
```

What do you need to dig a hole?

Thank you to Sean Yaw (the goat) for the analogy

What do you need to dig a hole?

What do you need to dig a hole?

	Pros	Cons
		
		
		

Thank you to Sean Yaw (the goat) for the analogy

What do you need to dig a hole?

	Pros	Cons
	• Cheap • Precise • No Training • Availability	• Slow • Labor
		
		

What do you need to dig a hole?

	Pros	Cons
	• Cheap • Precise • No Training • Availability	• Slow • Labor
	• Fast • Labor	• Expensive • Training
		

Thank you to Sean Yaw (the goat) for the analogy

What do you need to dig a hole?

	Pros	Cons
	• Cheap • Precise • No Training • Availability	• Slow • Labor
	• Fast • Labor	• Expensive • Training
	• Really good at digging	• Takes up a lot of garage space

What do you need to dig a hole?

	Pros	Cons
	• Cheap • Precise • No Training • Availability	• Slow • Labor
	• Fast • Labor	• Expensive • Training
	• Really good at digging	• Takes up a lot of garage space

Best tool for the job?

Burying your pet goldfish

Thank you to Sean Yaw (the goat) for the analogy

What do you need to dig a hole?

	Pros	Cons
	• Cheap • Precise • No Training • Availability	• Slow • Labor
	• Fast • Labor	• Expensive • Training
	• Really good at digging	• Takes up a lot of garage space

Best tool for the job?

Building Express tunnel to Bridger Bowl

Thank you to Sean Yaw (the goat) for the analogy

What do you need to dig a hole?

	Pros	Cons
	• Cheap • Precise • No Training • Availability	• Slow • Labor
	• Fast • Labor	• Expensive • Training
	• Really good at digging	• Takes up a lot of garage space

Best tool for the job?

Creating the foundation for a house

Thank you to Sean Yaw (the goat) for the analogy

What do you need to dig a hole?

	Pros	Cons
	• Cheap • Precise • No Training • Availability	• Slow • Labor
	• Fast • Labor	• Expensive • Training
	• Really good at digging	• Takes up a lot of garage space

Best tool for the job?

Digging a Well for water

Thank you to Sean Yaw (the goat) for the analogy

What do you need to dig a hole?

Best tool for the job?

Digging a Well for water

What do you need to dig a hole?

We can't use the best tool for the job unless we know that tool exists!

Best tool for the job?

Digging a Well for water

What do you need to dig a hole?

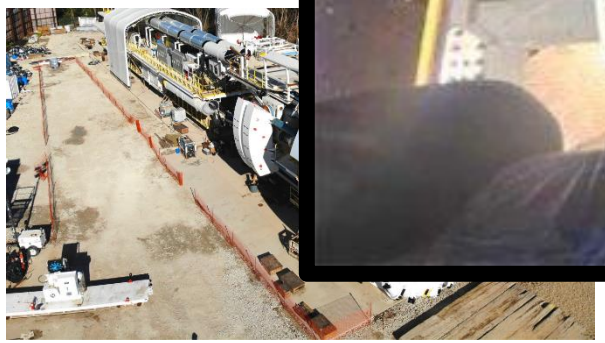
	Pros	Cons
	• Cheap • Precise • No Training • Availability	• Slow • Labor
	• Fast • Labor	• Expensive • Training
	• Really good at digging	• Takes up a lot of garage space

Best tool for the job?

Creating the foundation for a house

Thank you to Sean Yaw (the goat) for the analogy

What do you need to dig a hole?

	Pros	Cons
		
		ve
		a lot of ace

Best tool for the job?

Creating the foundation for a house

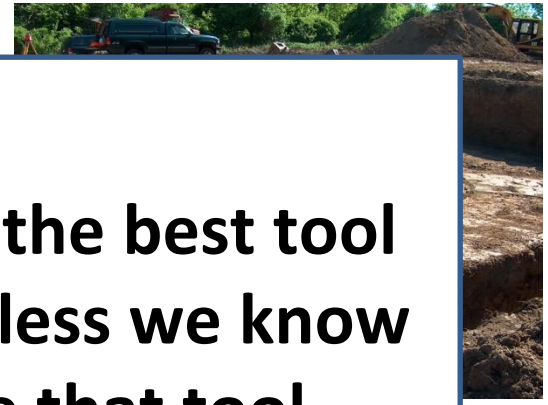
Thank you to Sean Yaw (the goat) for the analogy

What do you need to dig a hole?

Pros	Cons
	• Slow • Labor

Best tool for the job?

Creating the foundation for a house

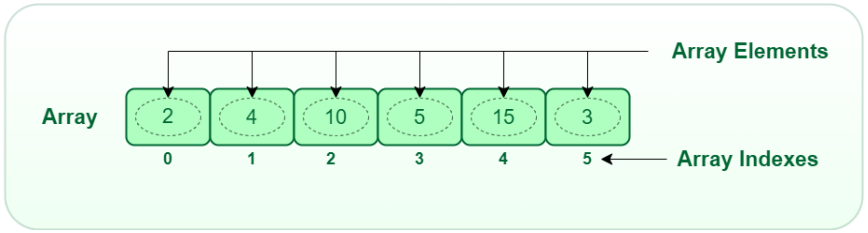


We can't use the best tool for the job unless we know how to use that tool

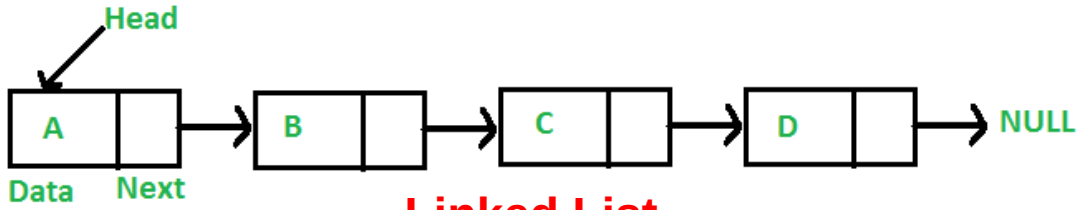
garage space

Thank you to Sean Yaw (the goat) for the analogy

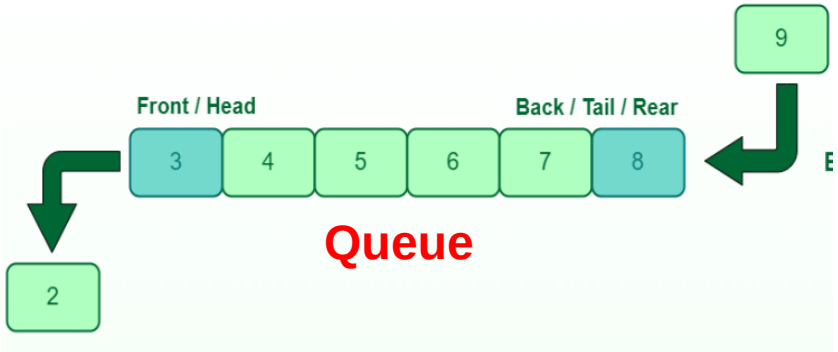
A **data structure** is a mechanism for storing and organizing data



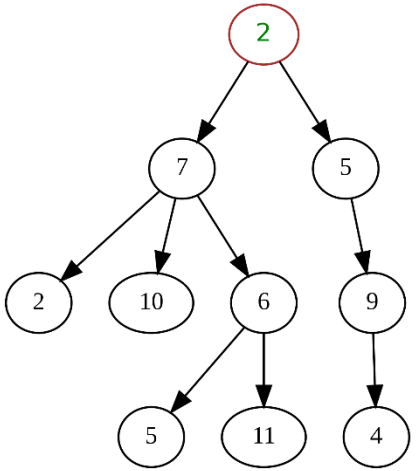
Arrays



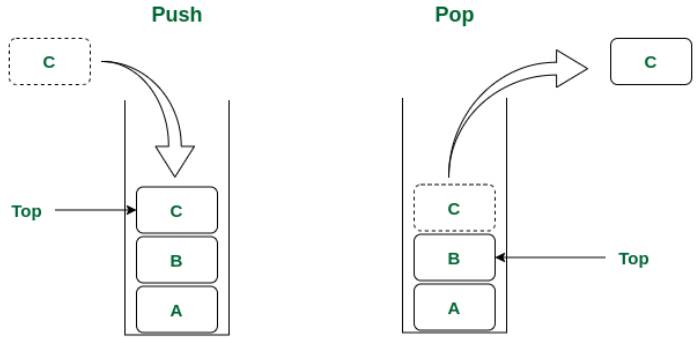
Linked List



Queue



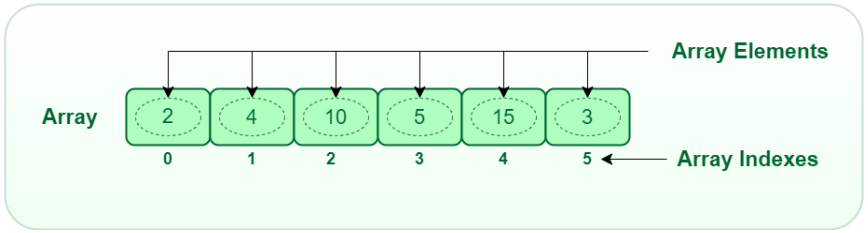
Trees



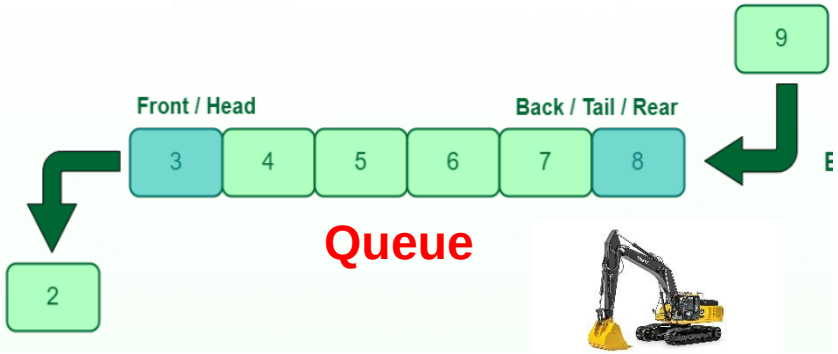
Stack

A **data structure** is a mechanism for storing and organizing data

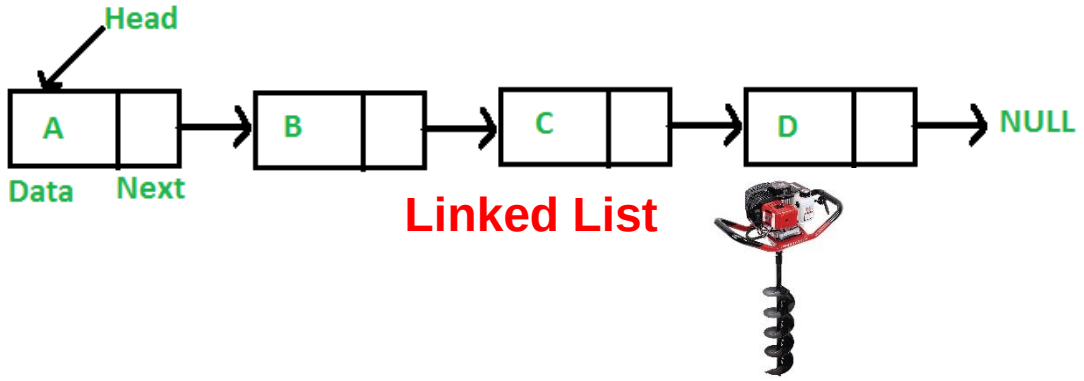
- We have structured ways of *accessing* and *managing* data



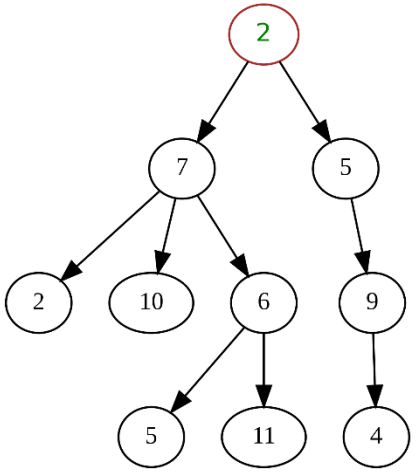
Arrays



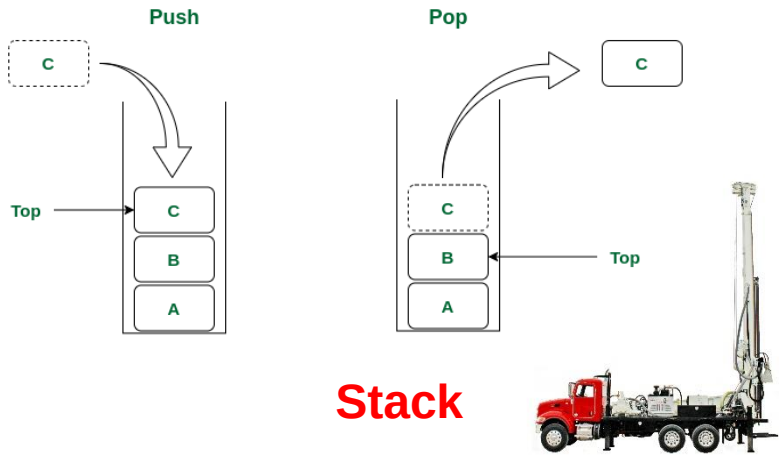
Queue



Linked List



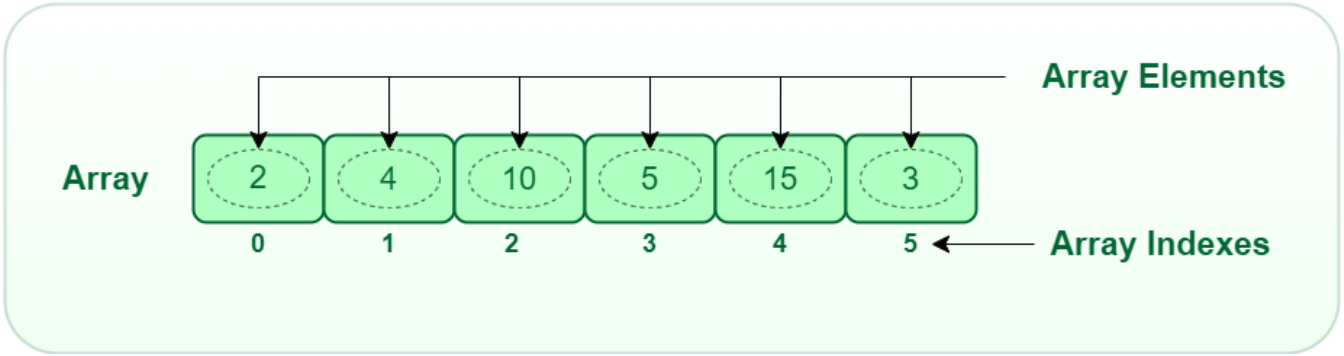
Trees



Stack

There are many types of data structure, and each data structure has its pros and cons

An **array** is a data structure that can hold multiple, similar values



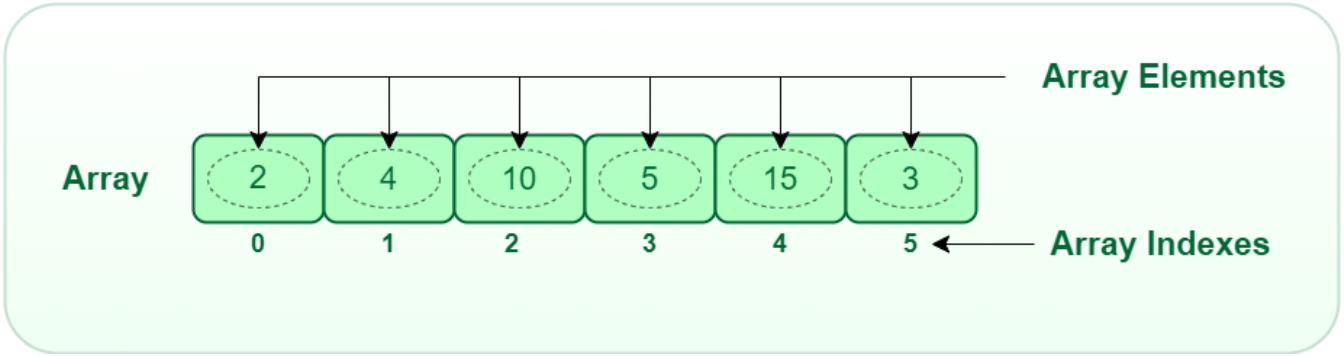

```
String[] cars = {"Volvo", "BMW", "Ford", "Mazda"};

int[] myNum = {10, 20, 30, 40};
```

Pros

- Holds multiple pieces of information
- Information is ordered (by index)
- Can easily change what is stored in each slot
- Can store duplicate data
- Easy to iterate through

An **array** is a data structure that can hold multiple, similar values




```
String[] cars = {"Volvo", "BMW", "Ford", "Mazda"};

int[] myNum = {10, 20, 30, 40};
```

Pros

- Holds multiple pieces of information
- Information is ordered (by index)
- Can easily change what is stored in each slot
- Can store duplicate data
- Easy to iterate through

Cons

- Can’t change the length
- Can only store one data type

Array Limitations

Cons

- **Can't change the length** *What can we do about this?*
- Can only store one data type

```
int[] myArray = {1, 2, 3};
System.out.println(Arrays.toString(myArray));
```

What if we wanted to add 4 to the array?

Array Limitations

Cons

- **Can't change the length** *What can we do about this?*
- Can only store one data type

```
int[] myArray = {1, 2, 3};
System.out.println(Arrays.toString(myArray));


int[] newArray = new int[myArray.length + 1];          // Create a new array that is one spot bigger
for(int i = 0; i < myArray.length; i++) {
        newArray[i] = myArray[i];                      // Fill new array with contents of old array
}
```

Array Limitations

Cons

- **Can't change the length** *What can we do about this?*
- Can only store one data type

```
int[] myArray = {1, 2, 3};
System.out.println(Arrays.toString(myArray));
```

```
int[] newArray = new int[myArray.length + 1];          // Create a new array that is one spot bigger
for(int i = 0; i < myArray.length; i++) {
        newArray[i] = myArray[i];                      // Fill new array with contents of old array
}
```

```
int new_value = 4;
newArray[myArray.length] = new_value;                  // add new value to array
myArray = newArray;                                    // Update reference variable
```

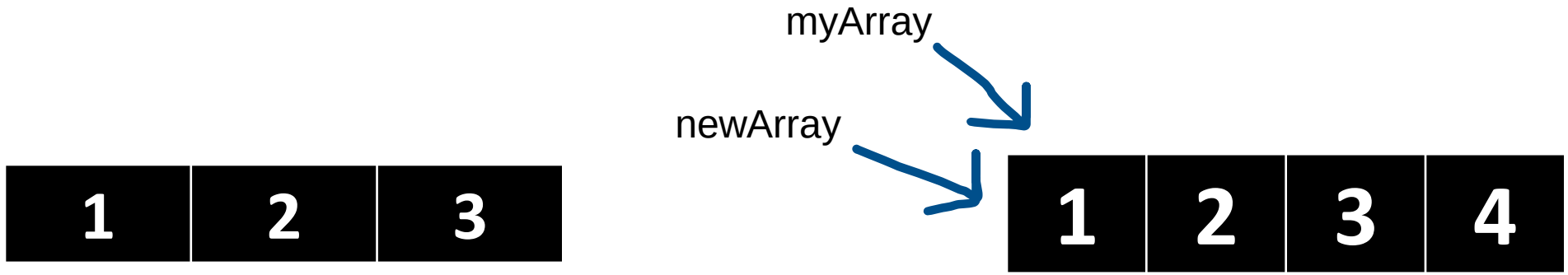
Array Limitations

```
int[] myArray = {1, 2, 3};
System.out.println(Arrays.toString(myArray));

int[] newArray = new int[myArray.length + 1];
for(int i = 0; i < myArray.length; i++) {
            newArray[i] = myArray[i];
}

int new_value = 4;
newArray[myArray.length] = new_value;
myArray = newArray;
```

We updated our reference variable (`myArray`) to point to our new array with the new element



Array Limitations

```
int[] myArray = {1, 2, 3};
System.out.println(Arrays.toString(myArray));

int[] newArray = new int[myArray.length + 1];
for(int i = 0; i < myArray.length; i++) {
            newArray[i] = myArray[i];
}

int new_value = 4;
newArray[myArray.length] = new_value;
myArray = newArray;
```

We updated our reference variable (`myArray`) to point to our new array with the new element

What happens to this array?
This is an unused object

Array Limitations

```
int[] myArray = {1, 2, 3};
System.out.println(Arrays.toString(myArray));

int[] newArray = new int[myArray.length + 1];
for(int i = 0; i < myArray.length; i++) {
            newArray[i] = myArray[i];
}

int new_value = 4;
newArray[myArray.length] = new_value;
myArray = newArray;
```

We updated our reference variable (`myArray`) to point to our new array with the new element

Java has a mechanism called **Garbage Collection**, with deletes unused object to free up memory

(this runs automatically!)

What happens to this array?
This is an unused object

Array Limitations

```
int[] myArray = {1, 2, 3};
System.out.println(Arrays.toString(myArray));

int[] newArray = new int[myArray.length + 1];
for(int i = 0; i < myArray.length; i++) {
            newArray[i] = myArray[i];
}

int new_value = 4;
newArray[myArray.length] = new_value;
myArray = newArray;
```

We updated our reference variable (`myArray`) to point to our new array with the new element

Java has a mechanism called **Garbage Collection**, with deletes unused object to free up memory

(this runs automatically!)

Java sees that we have an used/unreferenced object, so it will delete it!

Array Limitations

Cons

- **Can’t change the length**

  Solution

  Create new array, copy everything over
  (this can be expensive ☹)

```
int[] myArray = {1, 2, 3};
System.out.println(Arrays.toString(myArray));


int[] newArray = new int[myArray.length + 1];
for(int i = 0; i < myArray.length; i++) {
            newArray[i] = myArray[i];
}


int new_value = 4;
newArray[myArray.length] = new_value;
myArray = newArray;
```

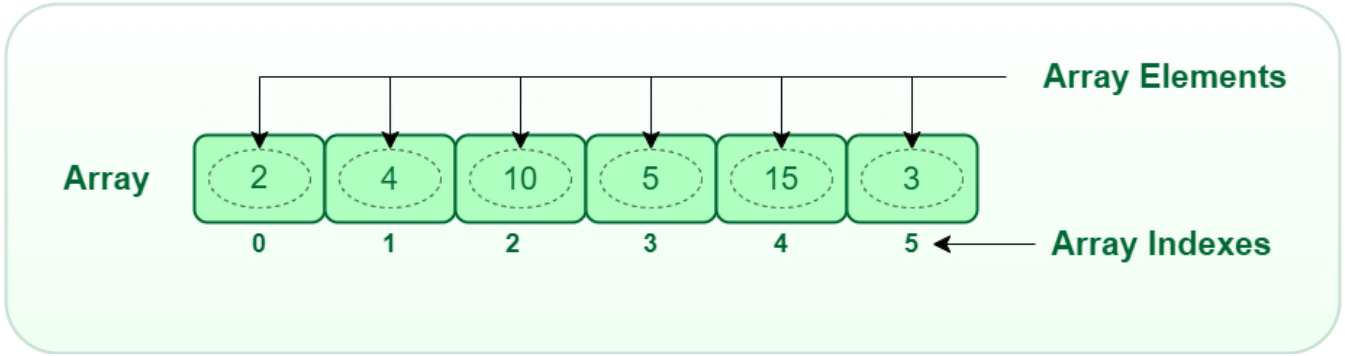
- **Can only store one data type**

  Solution

  Store an object, use two separate arrays, use a different data structure

An **ArrayList** is a data structure that can hold multiple, similar values (just like an array), BUT

- Dynamic, can easily resize
- Can easily add new elements and remove elements
- Like a Python list ☺



Somebody took `arrays`, and made them better

- Still have indices
- Still can only store one data type

Java **ArrayLists**

We first need to remember to import it ☺

```
import java.util.ArrayList;
```

Creating a new ArrayList

```
ArrayList<String> mylist = new ArrayList<String>();
```

We must specify what datatype the ArrayList will hold

Reference Variable

We never have to specify the size, because we can now easily grow/shrink our ArrayList!

Java **ArrayLists**

We first need to remember to import it ☺

```
import java.util.ArrayList;
```

Creating a new ArrayList

```
ArrayList<String> mylist = new ArrayList<String>();
```

We can add stuff to the ArrayList using the `.add()` method (built in method!)

```
mylist.add("Jack");
```

Java **ArrayLists**

We first need to remember to import it ☺

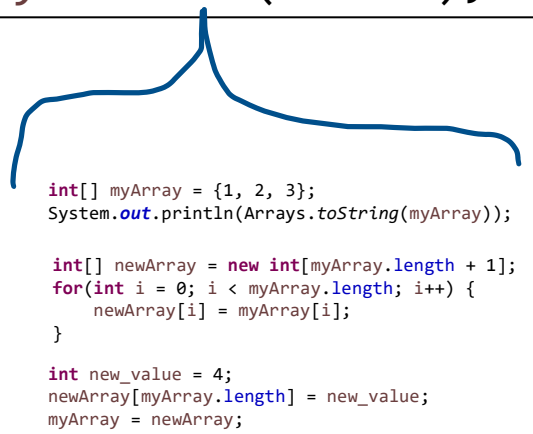
```
import java.util.ArrayList;
```

Creating a new ArrayList

```
ArrayList<String> mylist = new ArrayList<String>();
```

We can add stuff to the ArrayList using the `.add()` method (built in method!)

```
mylist.add("Jack");
```



```
int[] myArray = {1, 2, 3};
System.out.println(Arrays.toString(myArray));

int[] newArray = new int[myArray.length + 1];
for(int i = 0; i < myArray.length; i++) {
    newArray[i] = myArray[i];
}

int new_value = 4;
newArray[myArray.length] = new_value;
myArray = newArray;
```

Under the hood, it is

1. Creating a new array
2. Copy old contents
3. Add new element at the end
4. Updating reference variable

Java **ArrayLists**

We first need to remember to import it ☺

```
import java.util.ArrayList;
```

Creating a new ArrayList

```
ArrayList<String> mylist = new ArrayList<String>();
```

We can add stuff to the ArrayList using the `.add()` method (built in method!)

```
mylist.add("Jack");
```

To access elements in the array, we use the .get() method (we cannot use the square bracket index [ ])

```
System.out.println(mylist.get(2)); // this will print the String at index 2
```

Java **ArrayLists**

We first need to remember to import it ☺

```
import java.util.ArrayList;
```

Creating a new ArrayList

```
ArrayList<String> mylist = new ArrayList<String>();
```

We can add stuff to the ArrayList using the `.add()` method (built in method!)

```
mylist.add("Jack");
```

To access elements in the array, we use the .get() method (we cannot use the square bracket index [ ])

```
System.out.println(mylist.get(2)); // this will print the String at index 2
```

We can remove stuff by index, or by searching for a specific element

```
mylist.remove("Eli");
mylist.remove(0);
```

Java **ArrayLists**

```
import java.util.ArrayList;
        public class ArrayListDemo {

        public static void main(String[] args) {
                ArrayList<String> mylist = new ArrayList<String>();

                mylist.add("Jack");
                mylist.add("Tory");
                mylist.add("Sam");
                mylist.add("Eli");

                System.out.println(mylist);
                System.out.println(mylist.get(2));

                mylist.remove("Eli");
                mylist.remove(0);

                System.out.println(mylist);
                System.out.println(mylist.isEmpty());
        }
}
```

Java **ArrayLists** Example

Let's write a program that will keep track of high scores on an arcade machine

Each entry will have the player name (String), and their score (Int)

The program should allow for

- Adding a new high score
- Removing a score
- Print out scoreboard
- Print out top N scores
- Search for score by name

And we must use an **ArrayList** to hold all this information!