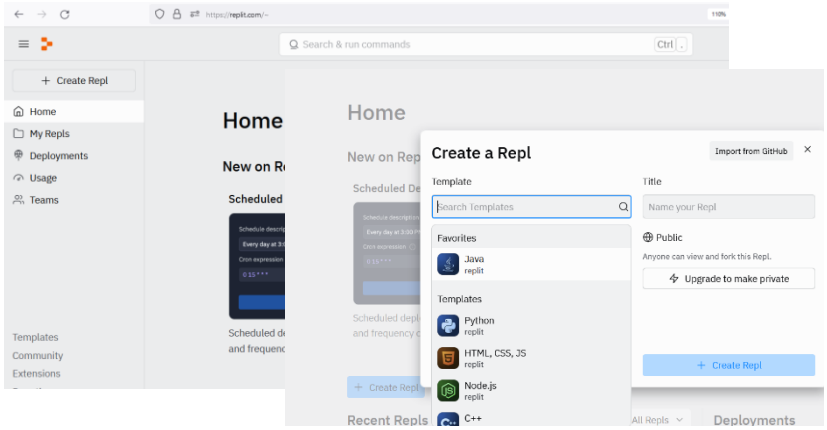
Good morning!

This QR code will take you to an application that introduces CS concepts, through Java. You can open it up on your cell phone or on a wider device, https://jacquikane.github.io/IntroJavaPocketBook/ - We will use the data structures card to see some visuals for the topic this morning.

Have you used repl.it before? REPL, at https://replit.com/ is often used in coding interviews. We will use it to demonstrate some code and concepts.



This discussion is about Linked Lists, so let's begin with the context.

Computing is all about solving problems, or manipulating data to achieve an objective. Data structures are computer programming building blocks that make it easier to build solutions using a programming language. You have been learning Python and now Java – what data structures can you think of from Python? List, tuple, dictionary ?

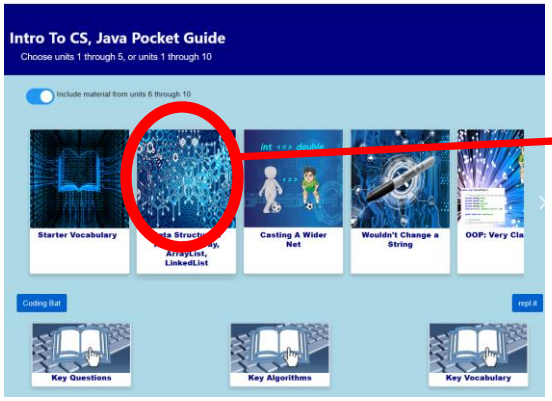
How about Java? What have you used so far in Java?

Today's discussion is about Linked Lists, and linked lists are another type of data structure in programming languages – like Python, JavaScript, Java, C, C++ ....

How about OOP – what is that? OOP, Object Oriented Programming, defines programming solutions based on chunks of information called objects. Objects are created from templates called classes. Think of an architectural blueprint as the class, and the many houses created based on the blueprint, as the objects.

Some programming languages have a predefined class to make it easy to instantly create a new object. In Java, this is the case with data structures that are used to model groups of information. The data structures we will look at today are going to involve classes and objects. Note, another

word for *object* is *instance*. We say that an object is *instantiated* from a class, or 'newed up' from a class.



Now let's look at that application, click on the image on the Data Structures card.

An array instance in Java is a data structure that can store units of information of the same type. The problem with an array is that it is of fixed size – you have to declare it to be of a certain number of elements, and if more elements become necessary, you are stuck.

Let's get into repl.it, and create an array in Java.

Create a new repl.it. In main, type in

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

 System.out.println(cars[2]);
```

You use an index to Access an array element, and we say they are zero based because the index of the first item is 0. The above print will print "Ford".

Now create another one using the new operator,

```
String[] perfumes = new String[4];
   perfumes[0] = "Penhaligon";
    perfumes[1] = "Gucci";
    perfumes[2] = "Atkinsons";
    perfumes[3] = "Dior";
    perfumes[4] = "Burberry";
```

What happens when the Burberry is added? This is what you see…

```
Exception in thread "main" java.lang.ArrayIndexOutOfBoundsException: Index 4 out of bounds for length 4
```

The array only stores the number of elements you declare it with, no more. So, what happens if you need to add more elements? Too bad!

An ArrayList data structure can take care of adding beyond any last element, it is implemented with an array as it's core – when items are added, the array is destroyed and another one is created. Does this seem efficient?

Here is code for an ArrayList. Note you have to import the code base into your code, with

```
import java.util.ArrayList;

 ArrayList<String> perfumeOils = new ArrayList<String>();
  perfumeOils.add("Lavender");
  perfumeOils.add("Bergamot");
  perfumeOils.add("Pine");
  perfumeOils.add("Cedar");
  perfumeOils.add("Rose");
  System.out.println(perfumeOils.get(perfumeOils.size()-1));
```

Can you guess what perfumeOils.size() – 1 is? This is the last element.

What happens if you are constantly adding/removing elements? Multiple core arrays are created and destroyed. This can be very inefficient, depending on the frequency.

Enter the LinkedList!

Let's see how the linked list is, visually, in the app.

The linked list is made of nodes, each of which has one or more references or pointers to another node – like a list of associations. It turns out that you can have..
- a singly linked list, in which each node points to only a previous or only a next node, or
- a doubly linked list in which each node has 2 pointers, one to the next node in sequence and the other to the previous.

This Linked List approach is much more flexible in terms of adding items, removing items.

Java's LinkedList class keeps the node implementation well hidden. Here is an example of using the LinkedList class in Java. You have to import it into your code, just like with ArrayList.

```
LinkedList<String> cookies = new LinkedList<String>();

  cookies.add("Chocolate");
```

```
cookies.add("Peanut Butter");
cookies.add("Raisin");
cookies.add("Snickerdoodle");
System.out.println(cookies.get(cookies.size()-1));
```

Behind the scenes, as well as the data, there are nodes, with pointers to the next and previous, as LinkedList is a doubly linked data structure. On the surface, it may just as well be an ArrayList.

Seeing as how we cannot examine the internals of a LinkedList object in Java, let's create a slimmed down minimal doubly linked list node, and linked list class. On the right, see how a singly linked list node might be created. *val* refers to the data the node contains.

The classes can include other methods, these are just the basics.

```
public class Node {
  public Node previous;
  public Node next;
  String val;

  public Node(String val) {
    this.val = val;
    this.previous = null;
    this.next = null;
  }

  public Node(String val,
          Node previous,
          Node next) {
  }

  public Node getPrevious() {
    return this.previous;
}

public Node getNext() {
   return this.next;
  }

  public String getVal() {
    return this.val;
  }
}
```

```
public class SinglyNode {
  public SinglyNode next;
  String val;

  public SinglyNode(String val) {
    this.val = val;
    this.next = null;
  }

  public SinglyNode(String val,
     SinglyNode next) {
  }

  public SinglyNode getNext() {
    return this.next;
  }

  public String getVal() {
    return this.val;
  }
}
```

If you include a toString method in Node, then you can do System.out.println(nodeObject);

```
public String toString() {
 return(“node : “ + this.data);
}
```

The nodes can be built up and linked together as is, but let’s create our own slim pseudo Linked List class framework (one that you can find problems with and solve yourself!), that we can use to create and demonstrate how the nodes are associated, and that can store the start and end of the list. BTW, blocks of code that belong to the object here, are called methods.

Many methods can be added to increase the usefulness and useability of our Linked List class. What are some tasks that the linked list should perform? How about adding an item, removing and item … for starters.

public class ShowyLinkedList {
 	
public Node head;
public Node tail;
 	
public ShowyLinkedList() {
this.head = new Node("0");
this.tail = this.head;
}
 	
}
	Declare the new class and add a constructor (special method that creates or allocates the space in memory to store the new object). You set the rules, you can make the first item 0.

public void add(String val) {
Node newNode = new Node(val);
this.tail.next = newNode;
newNode.previous = this.tail;
this.tail = newNode;
}
	What about method add … to add an item to the end of the list – can you draw a diagram illustrating this?

public void addAfter(int val, int index) {
// Create a new node
Node newNode = new Node(val);
// Find the node at the index
Node currentNode = this.head;
int i = 0;
while (i < index && currentNode.next != null) {
currentNode = currentNode.next;
i++;
}
// tail??
if (currentNode.next == null)
this.add(val);
else {
	A method that adds an item after a specific index

Node nodeAfter = currentNode.next;
currentNode.next = newNode;
newNode.previous = currentNode;
newNode.next = nodeAfter;
nodeAfter.previous = newNode;
}
}	

public Node removeAt(int index){
Node currentNode = this.head;
int currentIndex = 0;
while (currentNode.next != null && currentIndex < index) {
currentNode = currentNode.next;
currentIndex++;
}
// if head
if (index == 0){
this.head = currentNode.next;
this.head.previous = null;
return currentNode;
}
else if (currentNode.next == null){
// if tail
this.tail = currentNode.previous;
this.tail.next = null;
return currentNode;
}
else {
currentNode.previous.next = currentNode.next;
currentNode.next.previous = currentNode.previous;
return currentNode;
}
}
	A method that removes an item from a specific index.
	
	Ca you draw a diagram to illustrate this?
	
	What would you do if the head/tail needs to be removed?

public void prettyPrint() {
Node currentNode = myList.head;
while (currentNode != null) {
System.out.println(currentNode.getVal());
currentNode = currentNode.next;
}
}
	Print the content!
	You could define method toString (returns String) in Node, and then simply do System.out.println(currentNode)

How would you use this ShowyLinkedList class?

ShowyLinkedList myList = new ShowyLinkedList();	New it up, add elements. Now do a prettyPrint!

myList.add("1");
myList.add("2");
myList.add("3");
myList.add("4");
myList.add("5");	

Node currentNode = myList.head;	Get a reference to the head of the list

while (currentNode != null) {
System.out.println(currentNode.getVal());
currentNode = currentNode.next;
}
	Iterate through the list…. from head to tail

currentNode = myList.tail;
while (currentNode != null) {
System.out.println(currentNode.getVal());
currentNode = currentNode.previous;
}
	.. from the tail to the head

myList.addAfter(7, "xxx");
currentNode = myList.head;
while (currentNode != null) {
System.out.println(currentNode.getVal());
currentNode = currentNode.next;
}
	Now add an element and check the content of the list

myList.removeAt(4);
currentNode = myList.head;
while (currentNode != null) {
System.out.println(currentNode.getVal());
currentNode = currentNode.next;
}
	Remove an element and check the content of the list

The Java LinkedList class has many more methods:
https://docs.oracle.com/javase/8/docs/api/java/util/LinkedList.html

Do you see how it is handy to use the Java LinkedList class? That way, you can use the linked node framework, and do not have to create the methods. Sometimes in software engineering, for a very specific need, you might have to create a tailored linked list.

Do you know what a Linked list is? Do you know why we use a Linked List?

Now let's finish with some questions

- What is a stack? A stack is a Last-In-First-Out data structure, like a stack of mini pancakes. The pancake on the top of the stack was the last one added, and the first one eaten. How would you implement a stack with a linked list?
- What is a queue? A queue is a First-In-First-Out data structure, like a queue of people waiting in line for a service. The first person in the queue is served first. How would you implement a queue with a linked list?
- What else can you use a linked list for? Many years ago(!), when Windowing Operating Systems were newer, I used a linked list to create an association between modal windows that were related to each user in a user interface – the user could navigate between these. What about a shopping cart? Address book? Students enrolled in a class? A course? Dogs in the Hamilton County Shelter? Entities from a database for processing?

Note: In Java, a collections infrastructure is defined as part of the language. Collection is a generic term for a data structure that can hold items, like ArrayList and LinkedList. The collections infrastructure has 3 parts: interfaces (these are like protocols, that have to be implemented), implementations (like ArrayList and LinkedList), and algorithms.