Question 1 What's your favorite tool or library for Android? Why is it so useful?
Answer: Picasso is my favourite library for loading and caching images from the web. There are other libraries for doing this work but Picasso is my favorite library due to following reasons:

1. It's really simple and easy and I really like how the Api is written.
2. It saves lots of Android code for downloading and caching images from server. If we use normal Android code for this then we have make and execute **AsyncTask** for downloading image and then store the result in bitmap and set bitmap to source and then write lot of code for caching image. We can do all these task by just writing only one line of code using picasso.
3. **Picasso.with(this).load(imageUrl).into(mImageView);**
4. We can also do the resizing, cropping, scaling of images by using very simple functions. It allows to use different image sizes for different screen and also helps in performance issue and also saves lot of memory.
5. We can also do the error handling by simply overriding the onError() method.
6. We can import this library by writing simple one line of code : **compile 'com.squareup.picasso:picasso:2.5.2**
7. We can also load the default image in picasso if there is some problem in downloading image.

Question 2 - You want to open a map app from an app that you're building. The address, city, state, and ZIP code are provided by the user. What steps are involved in sending that data to a map app?

Answer:

1. Make an query URI for Strings provided by user. Make sure that string should be encode i.e. there should be %20 for space.
2. Make an Intent with Action_View.
3. Set the package name to com.google.android.apps.maps
4. startActivity with this intent.

Code:

```
private void launchMap(String address,String city,String state,String zipcode){
        Uri mapUri = Uri. parse ( "geo:0,0?q=" +address+ " " +state + " " + city + " " +zipCode);
        Intent mapIntent = new Intent(Intent. ACTION_VIEW , mapUri);
        mapIntent.setPackage( "com.google.android.apps.maps" );
        startActivity(mapIntent);
 }
```

Question 3 - Implement a method to perform basic string compression using the counts of repeated characters. For example, the string aabccccccaaa would become a2b1c5a3. If the "compressed" string would not become smaller than the original string, your method should return the original string. The method signature is: “public static String compress(String input)” You must write all code in proper Java, and please include import statements for any libraries you use.

Answer:

Code:

```
package com.udacity.dryrun;

import java.util.Scanner;

public class CompressStringClass {
	public static void main(String[] args){

		String s, output;
		Scanner in = new Scanner(System.in);
		System.out.println("Enter a string");
		s = in.nextLine();
		in.close();
		output = compressString(s);
		System.out.print(output);
	}

	public static String compressString(String str) {
		if (str.isEmpty()) {
			return "";
		}

		char[] chars = str.toCharArray();
		StringBuilder builder = new StringBuilder();

		int count = 1;
		char prev = chars[0];
		for (int i = 1; i < chars.length; i++) {
			char current = chars[i];
			if (current == prev) {
				count++;
			} else {
				builder.append(prev).append(count);
```

```
                    count = 1;
                }
                prev = current;
            }
            return builder.append(prev).append(count).toString();
        }
}
```

Question 4 - List and explain the differences between four different options you have for saving data while making an Android app. Pick one, and explain (without code) how you would implement it.

Answer:

1. **Shared Preferences** - Store private primitive data in **keyvalue** pairs. If our saving data is in key value pairs then we use SharedPreferences.
2. **Sqlite Database** - Store structured data in private database. If we have large amount of data then we can make databse. We can operate CRUD operations in our database.
3. **Internal Storage** - Internal storage used to store private data in device memory. We can store data in form of files in Internal Storage. This is private storage and other applications won't be access that data. It will use FileInputStream and FileOutputStream to save and fetch data in device memory.
4. **External Storage** - Store public data in shared external storage. If we want to save some data which is public user can access easily. This can be removable storage media (SD card) or internal storage. For this we have to ReadWrite permission from the user by declaring permission in manifest file.

SharedPreferences :
SharedPreferences class provides a generic framework.

To get a SharedPreferences object for your application, we can use one of two methods:

- **getSharedPreferences ()** Use this if you need multiple preferences files identified by name, which you specify with the first parameter.
- **getPreferences()** Use this if you need only one preferences file for your Activity. Because this will be the only preferences file for your Activity, you don't supply a name.

To write values:

1. Call **edit()** to get a **SharedPreferences.Editor** .
2. Add values with methods such as **putFloat()** and **putString()** . We use a key for saving the value. This key will be used when we read values from preferences later.
3. Commit the new values with **commit()** function.

To Read Values:

1. Restore the SharedPreferences object by using the same pref name that we have used at the time of saving.
2. Retrieve values by using the same key name that we used at the time of saving by using function **getFloat()** and **getString()** method. If there is no key found then we can return the default value.

Question 5 - What are your thoughts about Fragments? Do you like or hate them? Why?

Answer:
A Fragment is a piece of an activity which enable more modular activity design. Fragment has its own lifecycle methods and layouts. Fragments ensure the usability and flexibility of design i.e. we can reuse the same UI on different screens. For example, same layout can be used in phones and tablets within multiple activities. We can also combine multiple fragments into one screen. If we have a news application then we can show its listing and details in same screen in tablets and in different screen in phones only by the use of fragments.

I like fragments due to following reasons:

1. Dealing with device form factor differences. We can combine multiple fragments into one activity to build multi pane layouts. Fragments can be created for each form factor with form factor specific UI details being the responsibility of each fragment. The activity is then free to delegate the UI responsibility to the appropriate fragment for the current form factor.
2. A fragment can be used like plugnplay device. You can add or remove fragments while the activity is running.
3. Easy to pass information or data between app screens by using fragments. Before fragments it was difficult to pass the reference(objects) by using the intents. We have to make that object serializable. Now making each screen a separate fragment this data passing headache is completely avoided. Fragments always exists within the context of an activity and can easily access that activity.
4. Fragments plays major role in making rich UI design. Some of the UI components like view pager, navigation panel uses the fragments. We can also make new good UI custom components by using the fragments.

Question 6 - If you were to start your Android position today, what would be your goals a year from now?

Answer: I would pick up the projects which are challenging and requires sound working knowledge of Android framework. I will put in my knowledge and efforts to build an app that is stable in most cases and has reach to millions. I want to be an expert who can work on any challenging project and deliver his best.