1. Intents
   1. URIs
      1. <https://www.raywenderlich.com/160019/android-intents-tutorial-2>
      2. <http://www.vogella.com/tutorials/AndroidIntent/article.html>
      3. <https://developer.android.com/guide/components/intents-filters.html>
   2. Intents are code objects that are often used to ask for an action from a component of an application, whether that be the same app that made the intent or a different one. Said in a different way, intents are something that the application plans to do. Common uses of intents include changing the screen that’s visible, performing action in the background (not seen to the user), and broadcasting data such that all installed applications are able to handle it.  
      Intents are extremely important because they are what initializes action. It allows parts of applications to work together to produce the desired output. Every Android developer needs to understand them because they are the primary method by which action is began; without intents, functionality would never be realized.
   3. The most important idea behind intents is that they do what they sound like: they provide an abstract form of a request to perform an action. Of course, intents are not the actions themselves. A part of the Intent object, however is the information regarding the action that the intent proposes to the system.

Another important idea regarding intents is that of implicit and explicit versions. Explicit intents direct their attention to specific pieces of applications; this is useful when specific, inflexible actions need to be performed. Implicit intents are quite different; though they specify an action, they do not set the app component that should be targeted. Instead, they send it to whichever application is *able* to perform the action; then, sometimes at the user’s discretion, an application would be chosen for use. This is especially useful when, for example, a link to a website has been clicked (because sometimes the user would want to open the link in Chrome instead of the default browser).

1. Themes
   1. URIs
      1. <https://developer.android.com/guide/topics/ui/look-and-feel/themes.html>
      2. <http://www.vogella.com/tutorials/AndroidStylesThemes/article.html>
      3. <https://www.tutorialspoint.com/android/android_styles_and_themes.htm>
   2. In Android app development and design, a theme is a set of style rules that goes over an application. It allows for much of the appearance of the app to be generalized such that a large portion of style attributes for an activity only need be written in the theme definition file. In this way, switching a theme in an application is analogous to switching a CSS file on a webpage: objects of specific types will appear according to the style elements in the theme. This versatility gives the advantage of not needing to go back and replace every single style element with a different value; only the file name that is referenced in the <activity> or <application> need be changed.

Themes are created by defining them in a file with the XML extension in the /res/values/ path. They are then implemented by adding the “android:theme”=

“@style/FileName” property to an <activity> or <application> element.

Google has some themes that it has used in its native applications throughout the years, with the most recent being related to material design. Theme objects can use inheritance to capitalize on what Google has laid for us as a foundation: setting the parent theme of a theme object to the material design theme simplifies matters immensely.

* 1. Themes are one of the greatest things to happen to Android design. This is because it allows for true generality and the flexibility of the definition of a design separate from its implementation. Combined with the pre-made themes made by Google, themes allow for much more efficient Android application programming by a long shot.

Inheritance within themes is excessively important, because it takes a bit of weight off the shoulders of designers by allowing them to build on top of what has been made in the past (whether that be by that designer themselves or by Google, for example).

1. ViewGroups
   1. URIs
      1. <https://code.tutsplus.com/tutorials/android-from-scratch-understanding-views-and-view-groups--cms-26043>
      2. <https://developer.android.com/guide/topics/ui/declaring-layout.html>
      3. <https://mobiforge.com/design-development/understanding-user-interface-android-part-1-layouts>
   2. ViewGroups are non-visible object containers for other Views and/or ViewGroup objects. Several layers of ViewGroups can be combined to create a hierarchy (or even a pseudo-directory) of Views. The main groups are as follows: LinearLayout, AbsoluteLayout, RelativeLayout, TableLayout, FrameLayout, and ScrollView.

The ViewGroup is an important tool mainly because it allows the designer to implement a hierarchy of Views, storing multiple views in one container and moving that container around as one object instead of as however many it contains.

I could employ the LinearLayout in an application by putting three images side-by-side (or on top of each other vertically). I could use AbsoluteLayout to position some buttons on a the screen at a fixed-pixel location; this would be hard-coded location and thus not as good as something like RelativeLayout, which I could apply by placing a text field relatively lower than a row of buttons, which would all be located relative to one another horizontally. These buttons would provide options for editing the text (such as increasing the font size, toggling underlining/italics, etc.).

TableLayout would be useful when making a calculator with Button objects, because most of the buttons would take up the same amount of space (table layouts make uniform tables with rows and columns). FrameLayouts would be useful if I were to design an image gallery application that, once an image is clicked on to be viewed, allows users to press arrow buttons to navigate through the photos (this is because FrameLayout is ideal for displaying one main View at a time, and organizing the other views in something like a stack). Finally, ScrollView would be good for when I am displaying some user settings for an application, because if there are a large amount of settings to potentially be selected, the user would need to be able to scroll through them to access them.

* 1. I had not heard of ViewGroups before this assignment, but now I am glad that I have. They help the designer to be able to much more efficiently organize the components of their application.  
     Android app designers should constantly keep in mind all the types of ViewGroups that are available for use to fully maximize the benefits of them. The possibilities that emerge from these six main ViewGroups are exciting, and keeping them in mind is the key to satisfying such excitement.