Putting it simply:

As the name suggests, it's the context of current state of the application/object. It lets newly-created objects understand what has been going on. Typically you call it to get information regarding another part of your program (activity and package/application).

You can get the context by invoking `getApplicationContext()`, `getContext()`, `getBaseContext()` or `this` (when in a class that extends from `Context`, such as the Application, Activity, Service and IntentService classes).

Typical uses of context:

- **Creating new objects**: Creating new views, adapters, listeners:

```
TextView tv = new TextView(getContext());
ListAdapter adapter = new SimpleCursorAdapter(getApplicationContext(), ...);
```

- **Accessing standard common resources**: Services like LAYOUT_INFLATER_SERVICE, SharedPreferences:

```
context.getSystemService(LAYOUT_INFLATER_SERVICE)
getApplicationContext().getSharedPreferences(*name*, *mode*);
```

- **Accessing components implicitly**: Regarding content providers, broadcasts, intent

```
getApplicationContext().getContentResolver().query(uri, ...);
```

- `View.getContext()` : Returns the context the view is currently running in. Usually the currently active Activity.
- `Activity.getApplicationContext()` : Returns the context for the entire application (the process all the Activities are running inside of). Use this instead of the current Activity context if you need a context tied to the lifecycle of the entire application, not just the current Activity.
- `ContextWrapper.getBaseContext()` : If you need access to a Context from within another context, you use a ContextWrapper. The Context referred to from inside that ContextWrapper is accessed via getBaseContext().

this - return self reference
getContext() - return Context
getActivity() - return Activity

## Context.

> Quote from original answer :

As the name suggests, its the context of current state of the application/object. It lets newly created objects understand what has been going on. Typically you call it to get information regarding another part of your program (activity, package/application)

## Activity

Activity is a Java code that supports a screen or UI. In other words, building block of the user interface is the activity. Activity class is a pre-defined class in Android and every application which has UI must inherit it to create window. Activity represents the presentation layer of an Android application, e.g. a screen which the user sees. An Android application can have several activities and it can be switched between them during runtime of the application.

> Note : Activity extends Context. Context not a Activity.

`getApplicationContext()` Application context is associated with the Applicaiton and will always be the same throughout the life cycle

`getBasecontext()` should not be used jst use Context instead of it which is associated with the activity and could possible be destroyed when the activity is destroyed

There are two types of Context:

**Application context** is associated with the application and will always be same throughout the life of application -- it does not change. So if you are using Toast, you can use application context or even activity context (both) because toast can be displayed from anywhere with in your application and is not attached to a specific window. But there are many exceptions, one exception is when you need to use or pass the activity context.

**Activity context** is associated with to the activity and can be destroyed if the activity is destroyed -- there may be multiple activities (more than likely) with a single application. And sometimes you absolutely need the activity context handle. For example, should you launch a new activity, you need to use activity context in its Intent so that the new launching activity is connected to the current activity in terms of activity stack. However, you may use application's context too to launch a new activity but then you need to set flag Intent.FLAG_ACTIVITY_NEW_TASK in intent to treat it as a new task.

Let's consider some cases:

MainActivity.this refers to the MainActivity context which extends Activity class but the base class (activity) also extends Context class, so it can be used to offer activity context.

getBaseContext() offers activity context.

getApplication() offers application context.

getApplicationContext() also offers application context.

For more information please check this link.

**getApplication**, **getApplicationContext**, **LoginActivity.this** and **getBaseContext**, they all offer reference to the context.

Now the thing confuses is the declaration of different contexts and their specific-usage. To make things simple, you should count two types of context available in the Android framework.

1. Application Context
2. Activity Context

**Application** context is attached to the application's life-cycle and will always be same throughout the life of application. So if you are using *Toast*, you can use application context or even activity context (both) because a toast can be raised from anywhere with in your application and is not attached to a window.

**Activity** context is attached to the Activity's life-cycle and can be destroyed if the activity's `onDestroy()` is raised. If you want to launch a new activity, you must need to use activity's context in its *Intent* so that the new launching activity is connected to the current activity (in terms of activity stack). However, you may use application's context too to launch a new activity but then you need to set flag `Intent.FLAG_ACTIVITY_NEW_TASK` in intent to treat it as a new task.

*Now referring to your cases:*

`LoginActivity.this` though its referring to your own class which extends Activity class but the base class (Activity) also extends Context class, so it can be used to offer activity context.

`getApplication()` though its referring to Application object but the Application class extends Context class, so it can be used to offer application context.

`getApplicationContext()` offers application context.

`getBaseContext()` offers activity context.

**Tips: Whenever you need to manipulate `Views` then go for *Activity-Context*, else*Application-Context* would be enough.**

**View.getContext():** Returns the context the view is currently running in. Usually the currently active Activity.

**Activity.getApplicationContext():** Returns the context for the entire application (the process all the Activities are running inside of). Use this instead of the current Activity context if you need a context tied to the lifecycle of the entire application, not just the current Activity.

**ContextWrapper.getBaseContext():** If you need access to a Context from within another context, you use a ContextWrapper. The Context referred to from inside that ContextWrapper is accessed via getBaseContext().

The `this` keyword and the context methods aren't really related, except that they both sort of define scopes of where things could be.

`this` is a Java Keyword is a reference to the current object. Since you can have nested classes, you can use Class.this to disambiguate which this you are referencing. For example, lets say we have this code:

```
1  public class MyActivity extends Activity {
2      private final int someValue;
3
4      public MyActivity(int someValue) {
5          this.someValue = someValue;
6      }
7
8      public class MyClickListener implements Vie
9          public void onClick(View view) {
10             Log.v(MyActivity.this.someValue);
11         }
12     }
13 }
```

Line 5 of that code uses `this` to identify that the someValue being assigned to is the member variable. The MyClickListener is an inner class, it exists in the context of an instance of MyActivity. Part of what it wants to do is access the someValue instance variable inside the activity. So it uses `MyActivity.this` to make it clear that is where the someValue variable should come from.

The contexts are Android specific scoping objects that extend the android.content.Context class. There is a sort of global context - the Application Context - which can be used to get global state, and then there are usually one or more smaller-scoped contexts which hold more specific state information (like the Activity). In general, you don't really need to worry about all the different levels and methods, you just need to get a Context somehow, usually the smallest scoped one, and use it. And that is usually by calling the Activity's getContext() method. The context itself will handle delegating or searching broader contexts if it doesn't have the information you need. So you generally don't need getBaseContext() (though if you were to write a Context implementation yourself, you might).

The ApplicationContext is used when you might need access to a context but you need to make sure it has a long enough life. For example, the Activity is a Context, and its getContext() method usually returns itself. But if you have a class which uses the context but sticks around for longer than the life of the Activity (for example it might be used in a Service or is something that gets passed from one Activity to the next) then using the Activity as a context isn't the best idea, it could cause memory leaks or unexpected behavior. So in those cases you need a context that lasts the lifetime of the application, and that would be what getApplicationContext() returns.

`this` refers to your current object. In your case you must have implemented the intent in an inner class ClickEvent, and thats what it points to.

`Activity.this` points to the instance of the Activity you are currently in.

**Context** is context of current state of the application/object.It™s an entity that represents various environment data . Context helps the current activity to interact with out side android environment like local files, databases, class loaders associated to the environment, services including system-level services, and more.

A Context is a handle to the system . It provides services like resolving resources, obtaining access to databases and preferences, and so on. An android app has activities. It's like a handle to the environment your application is currently running in. The activity object inherits the Context object.

It lets newly created objects understand what has been going on. Typically you call it to get information regarding another part of your program (activity, package/application).

** UPDATE: Android Complete tutorial now available here.

### According to Android Documentation

Interface to global information about an application environment. This is an abstract class whose implementation is provided by the Android system. It allows access to application-specific resources and classes, as well as up-calls for application-level operations such as launching activities, broadcasting and receiving intents, etc.

### Different invoking methods by which you can get context

1. getApplicationContext(),

2. getContext(),

3. getBaseContext()

4. or this (when in the activity class).

### Examples for uses of context:

1. **Creating New objects:** Creating new views, adapters, listeners

```
TextView tv = new TextView(getContext());
ListAdapter adapter = new SimpleCursorAdapter(getApplicationContext(), ...);
```

2. **Accessing Standard Common Resources**: Services like LAYOUT_INFLATER_SERVICE, SharedPreferences:

```
context.getSystemService(LAYOUT_INFLATER_SERVICE)
getApplicationContext().getSharedPreferences(*name*, *mode*);
```

3. **Accessing Components Implicitly**: Regarding content providers, broadcasts, intent

```
getApplicationContext().getContentResolver().query(uri, ...);
```

You will need the Context class is when creating a view dynamically in an activity. For example, you may want to dynamically create a TextView from code (First Example). To do so, you instantiate the TextView class. The constructor for the TextView class takes a Context object, and because the Activity class is a subclass of Context, you can use the this keyword to represent the Context object.
A Context object provides access to the application's resources and other features. Each Activity is a Context

and each View needs a Context so it can retrieve whatever resources it needs (including things like system-defined resources).

### Difference between Activity Context and Application Context:

They are both instances of Context, but the application instance is tied to the lifecycle of the application, while the Activity instance is tied to the lifecycle of an Activity. Thus, they have access to different information about the application environment.

If you read the docs at getApplicationContext it notes that you should only use this if you need a context whose lifecycle is separate from the current context.

But in general, use the activity context unless you have a good reason not to.

### Need of Context :

The documentation says that every view needs the context to access the right resources (e.g. the theme, strings etc.).

But why in the constructor and not through setContentView(View)?

1.Because the resources must be accessible while the view is being constructed (the constructor will need some resources to fully initialise the view).

2.This allows the flexibility of using a context that is different from the one of the current activity (imagine a view that uses some other string resources and not the ones from the current activity).

3.The designers of the Android SDK seem to have chosen that the context must be set only once and then stay the same throughout the lifetime of the view.

### Why context is not determined automatically at construction point?

1.Because there exists no static variable that would tell you the current global context of your application. The method getApplicationContext() is the closest to this, but it’s not static, so you need an instance of the Activity object to call it.

2.The Java language provides an option to look through the call stack and find whether the View has been constructed in a Context class. But what if there are many? Or what if there are none? This method is very expensive and error prone. So the designers of the API decided that a context must be manually provided.