From Django.Dispatch import receiver

What does it accomplish?
Django's dispatch is used to notify the application when an action is taken somewhere in the application. Receiver in our application is called when users update their profile or when a new profile is created. The parameters for this function is a signal and keyword arguments to connect.

How does this technology accomplish what it does?
This function takes in a signal and keyword argument to connect the receiver to signals. Specifically, the receiver is connecting signals to a specific function, in our case it's used to connect a signal post_save with the user as the sender to our function create_or_update_user_update in /authentication/models.

- As we can see here**,** receiver is used to call a function when a request is made or a signal is given. Django, as described here, provides built-in signals which allows users to receive a signal when specific actions are called.
    - In our case the signal is "post_save" which is a signal from django's built in signals.
    - This specific signal is when users save their changes onto the profile or create a new profile
- A signal in django are implicit function calls

From django.apps import AppConfig

What does it accomplish?
As explained here, when Django needs to interact with the installed application we need AppConfig to hold each data for each application. For our project, we use AppConfig for both apps.py in both the "Blog" and "Authentication" folders. Thus, we are creating an instance of each application for our blog (where we hold posts and interact with users) and authentication, which verifies users are signed up with in account.

How does this technology accomplish what it does?
When we create a pluggable app (blogs or authentication), we create a class (whatever we want to call it) and pass in AppConfig. Within this class we call name and assign the name of the app we want to reference. As seen here, when we assign 'blog' to the name we are really changing the of AppConfig. Doing so, allows us to store our data for the 'blog' application and lets Django know which application to load.

From django.contrib import admin

What does it accomplish?
When we use this specific import, we are creating an admin page for each of our model (which is the view that we create in our model.py function for each aspect in our project, this includes blog, direct messages and authentication). This admin page allows us developers to manage site and it's data. For example, you could update a user's account.

How does this technology accomplish what it does?
As described here, when we import admin we are accessing django's standardized admin page. The function admin.site.register(Post) is assigning Django's standard admin page ours Posts. Specifically, for this function we need to pass a model view so the data we are analyzing is our posts.

From django.db.models.signals import post_save

What does it accomplish?
As mentioned above, Django, uses signals to communicate within an application. Post_save is a signal that is sent after the Django's model save() function is ran. The post_save function is passed to a receiver, which dictates when the following function is ran. In authentication/model.py, after Profile is saved, a post_save signal is sent to the receiver to let the following function be ran.

How does this technology accomplish what is does?
There is a built-in save function within each model from django.db as we can see here. When we create our model, in authentication/model.py for Profile, we pass a model for the class. At the end of this class the built in save() function is called. Following this class, we see a receiver function which is taking in post_save. The signal is waiting until after the original save() function is called in our Profile class to send the signal for our receiver function. After we receive this signal, the function under receiver is ran.

From django.contrib.auth import views as auth_views

What does it accomplish?
This import is used for our urls in urls.py for our class_project folder. In this file, we are taking one of the provided views created by Django and we are customizing by assigning the view (auth_views) to follow the template that we create. For example, we are using the template name “authentication/Login.html,” so we are building upon auth_view for our login.

How does this technology accomplish what it does?
As described here, Django has an authentication system that handles authentication and authorization. As we see here, Django provides us with several views that we can use for handling login, logout, and password management. However, we need to build upon these views by passing in our own .html templates through the url patterns (like here). As we see here django.contrib.auth provides us different views like the class LoginView. We can override the default html template for login for LoginView in our “authentication/Login.html” page by using path(‘login/’, auth views.LoginView.asview(template_name=’authentication/Login.html’),name=’login’)

From django.contrib import messages as messages1

What does it accomplish?
We are using this specific import to send users a notification message, like a pop up message. In our case, we are using django.contrib import messages to display our message1, to let users know that their account has been updated in our profile function in “blog/views.py” if they have successfully updated their profile page.

How does this technology accomplish what it does?
As we see here, we can update the message we want to display after a success profile update by doing messages.success(request, f’Account Successfully Updated!’). Messages are ran in Django using a middleware class and a corresponding context processor . A middleware class as described, allows us to attach a message after a certain request is completed. The request in messages.success() is after a user updated valid information within our POST request to update their profile. The context processor of our message, is the information that we are associating with our message. In this case, we override the content by providing info_form, picture_form, and unread to be associated with our message.