Linking to Objects from Django Templates

When working with Django Templates, you will often find yourself needing to link directly from a template to another url. For example, linking from a list of posts to a specific post, or linking from a post to a category page, or from the profile of one user to another.

Since in Django, all data will be backed by a model, most links to pages can be thought of as linking to *views* on a particular model. For example, linking to the detail page of a post is linking a *view* that renders the details of that post to a template. In fact, you can think of this link as linking directly to the post object itself. Linking to a user's profile page is really linking to that instance of the User model, and the view simply controls how it is shown.

Anyway, I'll stop rambling abstractly and explain concretely using the post_list and post as an example. There are at least three ways to link from the lists of post directly to a post. The first is bad but easy, the second is better (though still not great) and harder, the third is more confusing, but the best and *right* way to do it.

1. **Hard coding the url into the template using template variables**
   This way is how the code in the lab does it. It is easy to do and to think about. Linking to a particular post, for example, is as easy as writing:

   ```
   <a href="/blog/posts/{{ post.id }}/true">
        {{ post.title }}
   </a>
   ```

   This will work. **BUT** imagine that you need to link to a post from more than one place. Then that URL ("/blog/posts/{{ post.id}}/true") will appear in many templates. What happens if you decide that you want to change how the URLs are structured? Let's say you decide that you don't want to have the '/blog' at the beginning anymore. Once you make the change in your urls.py, you still have to go through **all your templates** and manually change all of the appearances of the old url to the new one. Besides leaving you open to bugs (if you miss one of the old urls) this is tedious, a waste of time, and for large websites, nearly impossible. There has got to be a better way.

2. **Using get_absolute_url and hardcoding the URL return value**
   This method of linking to an object requires you to add a method to the model of the object that you want to change (note… you do not have to rerun syncdb after you add methods). That method is `get_absolute_url,` and should return the URL that identifies that particular object. For our Post model, we would add the following code:

```
def get_absolute_url(self):
     return "/blog/posts/%i/true" % self.id
```

   Then, to link to a post from within a template, you would write

```
<a href="{{ post.get_absolute_url }}">
     {{ post.title }}
</a>
```

   This way, when you change the url mappings, you only have to make the change in one place and then all of the links in your templates will work. Note that you do not *have to* use the name get_absolute_url. However, the admin interface and other tools look for this method, so if you are going to use this method of linking to objects, you should call your method get_absolute_url.

   This technique is much much better than the first technique, but the url is still hardcoded somewhere. If there are a lot of models with hardcoded urls, it is easy to forget to change one when changing around a url structure, which could lead to sneaky bugs appearing in your website. If you are interested in an **even better** way to do it, read on.

3. **Using get_absolute_url and the models.permalink decorator**
   The technique showed above (2) is much better than the first technique. But the url is still hard-coded. With this, the people working on urls.py files will have to make sure that they don't forget to also change the return values of all affected get_absolute_url methods. This leaves the developers open to mistakes and could introduce bugs.

   A way around this is the models.permalink *decorator*. What is a decorator? It's an advanced python topic that is easy to use but hard to create. Don't worry about what it is – if you are curious, google it. Or ask one of the instructors.

   Ok – but here's one way to use it. In urls.py, add a 'name' keyword argument to the url that maps to your model, setting the name to something identifiable. For example, in blog/urls.py:
   Change the line:

```
url(r'^posts/(?P<id>\d+)/((?P<showComments>.*)/)?$',
     'blog.views.post_detail'),
```

   to:

```
url(r'^posts/(?P<id>\d+)/((?P<showComments>.*)/)?$',
     'blog.views.post_detail', name='post_detail'),
```

   Then, in the Post model, replace your `get_absolute_url` method with the following code:

```
@models.permalink
def get_absolute_url(self):
     return ('post_detail',(),
               {'id':self.id,'showComments':'true/'})
```

   The models.permalink decorator will automatically find the correct regex and generate the correct URL using the parameters and keyword parameters that you return.