# Dynamic Addressing

Sometimes it's not very clear how many URLs you need to define for your project, that's where Dynamic Addressing comes in handy. By defining a function that handles all of the needed URLs which might be generated later on.

**Let's begin**

1. Make a view function as we did before.

```
from django.http import HttpResponse

def dynamic_view(request, dynamic):
    return HttpResponse(dynamic)
```

1. Bind the URL to a path DYNAMICALLY, as follows.

```
from django.urls import path
from . import views

urlpatterns = [
    path('<dynamic>', view=views.dynamic_view),
]
```

- The `dynamic` argument will be fed into the `dynamic_view()` method with the same name. (`dynamic`)
- Now, anything the client types in the URL, if is not already binded to another view, will bond to this dynamic view.

1. Test

Type any URL and see what happens: `localhost:8000/MyOwnURL`

## Multiple Dynamic URLs

The good thing about Dynamic Addressing is that you can have as many Segments as you need. you just need to update your files in this way:

```
from django.http import HttpResponse

def dynamic_view(request, dynamic, dynamic2, dynamic3):
    return HttpResponse((dynamic, dynamic2, dynamic3))

from django.urls import path
from . import views

urlpatterns = [
    path('<dynamic>/<dynamic2>/<dynamic3>', view=views.dynamic_view),
]
```

## Type Annotation for Dynamic Addressing

There also this feature to control which data type the client can put in the URL. This feature is most useful when you want to control which function must be called based on Data Type.

```
urlpatterns = [
    path('<int:dynamic>', view=views.another_view),
    path('<str:dynamic>', view=views.dynamic_view),
]
```

**Note that you must put less general data types first.**

Because django tries to cast the URL segment to other data types along with some exception handling. Anything can be converted to string but that doesn't check out for int.

## HttpResponseRedirect

Among all this Dynamic Addressing crap, one cool thing is Redirecting. This is how you can redirect the client to URL B if URL A was called. **Note that the status code for Redirection will be in range 300.**

```
from django.http import HttpResponseRedirect

def my_view(request):
    return HttpResponseRedirect("path/to/url/B")
```

## URL names

Let's learn some best practice about URLs= and redirecting. In a relatively large project, there will be a bunch of URLs, many of them redirecting to each other with something like above.

If you were to change some tiny part of a URL (I mean the path/to/url part), there will be CONSEQUENSES!

To resolve this like a good programmer, Django has the feature of naming urls in `path()` method. You can simply add a keyword argument `name="unique_name"` and then, by using `reverse()` method in the view in which you used a redirection, you can read the URL dynamically.

```
urlpatterns = [
    path('<int:dynamic>', view=views.another_view),
    path('<str:dynamic>', view=views.dynamic_view,
name="unique_name"),
]

from django.urls import reverse

def my_view(request):
    return HttpResponseRedirect(reverse("unique_name", args=[]))
```

`args` is the list of all segments that must appear after the url to which `"unique_name"` is refrencing.

By using this feature, you are sort of storing your URL inside a variable (so to speak) that will look for anytime the URL is called. Will this affect the websites performance? Of course not! it's just like calling a variable by reference.