


Open   Opened 2 days ago by  **Rubén Montero**

# Las tablas del 1 al 10

## Resumen

- Repasaremos qué son los *path params* y los veremos en Django
- Crearemos una nueva función en `endpoints.py` que producirá como resultado una lista JSON con los valores de una tabla de multiplicar
  - Recibirá un parámetro, y en función de si vale `one`, `two`, `three`, ... el resultado será la tabla del **1, 2, 3**,... hasta **10**
  - Devolverá un `404` si el valor del *path param* es desconocido

## Descripción

Tener un *endpoint* para la tabla de multiplicar del **6** no está mal, pero sería mejor poder ver **más** tablas de multiplicar.

Para ello, comenzaremos a trabajar con ***path params*** en Django.

Ya sabemos que un ***path param*** es una parte variable de la URL que se encuentra en la parte principal de la dirección. Por ejemplo:

- [https://es.wikipedia.org/wiki/Isaac\\_Newton](https://es.wikipedia.org/wiki/Isaac_Newton)
- [https://es.wikipedia.org/wiki/Michael\\_Schumacher](https://es.wikipedia.org/wiki/Michael_Schumacher)
- <https://twitter.com/elonmusk/media>
- <https://twitter.com/neiltyson/media>

...son ejemplos de URLs con *path params*.

Podemos *señalar* dichos parámetros entre *mayor que* y *menor que* ( `<` `>` ), por ejemplo:

- `https://es.wikipedia.org/wiki/<article>`
- `https://twitter.com/<user>/media`

## ¿Y esto qué tiene que ver con nuestro **SimpleAPI** ?

Vamos a intentar que las siguientes URLs:

- <http://localhost:8000/multiplication/one>
- <http://localhost:8000/multiplication/two>
- <http://localhost:8000/multiplication/three>
- <http://localhost:8000/multiplication/four>
- <http://localhost:8000/multiplication/five>
- <http://localhost:8000/multiplication/six>
- <http://localhost:8000/multiplication/seven>
- <http://localhost:8000/multiplication/eight>
- <http://localhost:8000/multiplication/nine>
- <http://localhost:8000/multiplication/ten>

Sean manejadas por nuestra API mediante un único *endpoint*:

- `http://localhost:8000/multiplication/<number>`

## ¿Cómo?

Muy fácil.

En `urls.py` añadiremos una línea así:

```
urlpatterns = [  
    # ...  
    path('multiplication/<number>', endpoints.multiplication_table),  
]
```

## ¿Se puede usar `<` `>` dentro de la URL?

Es **precisamente** la forma de indicar en Django que esa parte es variable. O sea, un *path param*.

Ahora falta definir la función `multiplication_table`, ¿no?

Correcto.

Abramos el fichero...

... `endpoints.py`

Allí añadimos la firma del método que acabamos de indicar, con **una condición muy especial**:

```
def multiplication_table(request, number):  
    # ...
```

¡Tiene un parámetro 'extra'!

- El *path param* debe añadirse como **parámetro** de la función
- El **nombre** del *path param* debe coincidir con el **nombre** del parámetro (en este caso, `number`)
- Si hubiera más de uno, se añade también

Entonces... ¿Ya está conectado?

Sí.

Puedes imaginarte que si tu método *fuera así*:

```
def multiplication_table(request, number):  
    print("El usuario ha visitado la tabla del " + number)  
    # ...
```

Entonces, al navegar a:

- <http://localhost:8000/multiplication/one>

...tu servidor haría el `print`:

```
El usuario ha visitado la tabla del one
```

Otro ejemplo. Al visitar:

- <http://localhost:8000/multiplication/three>

...tu servidor haría el `print`:

```
El usuario ha visitado la tabla del three
```

## La tarea

Se pide que en tu función `multiplication_table` tengas varios `if` anidados para manejar los siguientes **posibles** valores del *path param*:

- `one`
- `two`
- `three`
- `four`
- `five`
- `six`
- `eight`
- `nine`
- `ten`

...y para cada uno, devuelvas una lista JSON con los valores de la tabla de multiplicar. Igual que hacíamos con la tabla del **6** en la tarea anterior.

Por ejemplo<sup>1</sup>:

```
def multiplication_table(request, number):  
    if number == "one":  
        return JsonResponse([1, 2, 3, 4, 5, 6, 7, 8, 9, 10], safe=False)  
    elif number == "two":
```

```

    return JsonResponse([2, 4, 6, 8, 10, 12, 14, 16, 18, 20], safe=False)
elif number == "three":
    # ...

```

(Como ves, puedes aprovechar la sentencia `elif` que equivale a un `else` y un `if`)

Si al llegar al final, el parámetro `number` no es ninguno de los considerados más arriba:

```

def multiplication_table(request, number):

    # ...

    elif number == "ten":
        return JsonResponse([10, 20, 30, 40, 50, 60, 70, 80, 90, 100], safe=False)
    else:
        # ¡No coincide con ninguno!

```

...entonces se devolverá una respuesta JSON con este contenido:

```

{
  "error": "Number invalid. Only one to ten are supported"
}


```

...y el código HTTP 404.

## Por último

Verifica que tu código pasa el `test` asociado a la tarea.

Haz `commit` y `push` para subir los cambios al repositorio.

1. Se muestra una sugerencia. Puedes escribir tu código tan compacto y legible como desees, usando técnicas diferentes si lo deseas (e.g. [map...](#)) 



[Rubén Montero @ruben.montero](#) changed milestone to [%Sprint 4](#) 2 days ago



[Ania Blanco @ania.blanco](#) mentioned in commit [276e860d](#) 7 hours ago