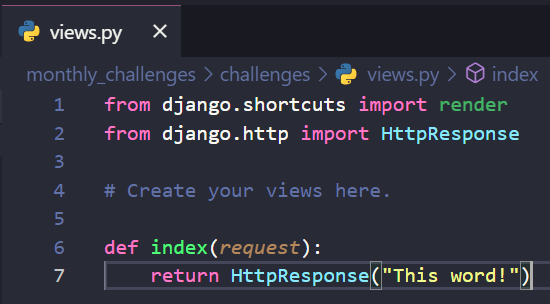
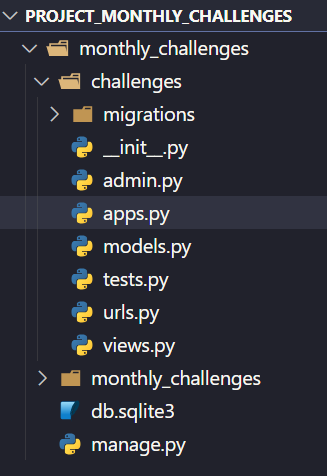
# Section 3: URLs & Views

## 21. Creating a First View & URL

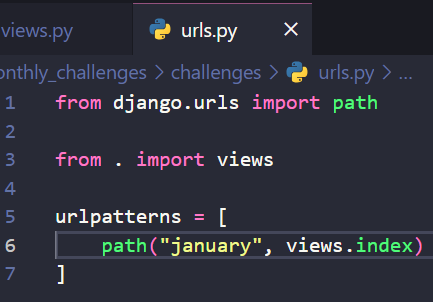
* View là thứ mà Django trả về cho người dùng khi nhận được request:
* Ví dụ về view trong bài này:



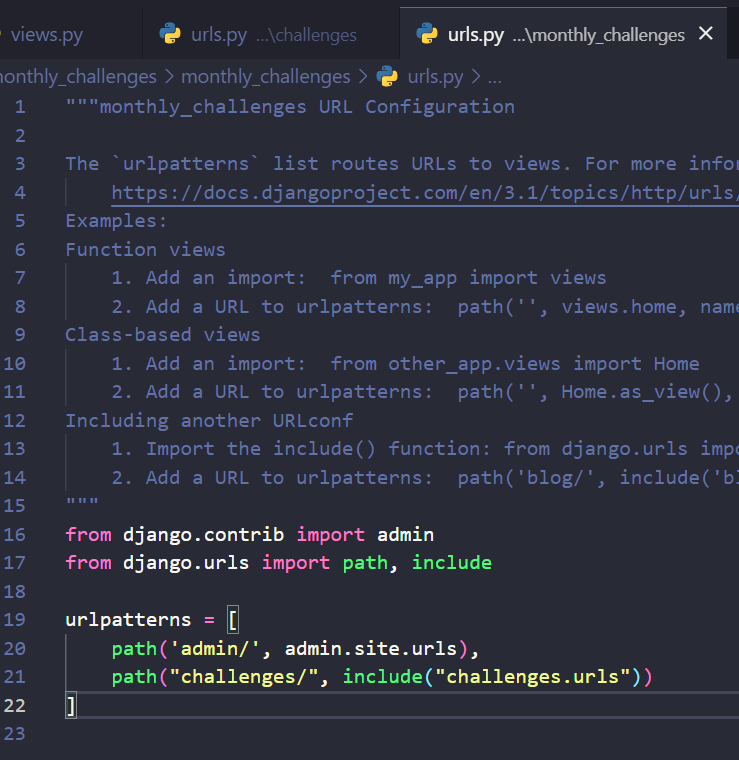
* Cần cho server biết là khi có đường dẫn nào thì cần trả về cái view này:
* Tạo ra 1 file là urls.py ở ngay trong thư mục của app “challenges”:



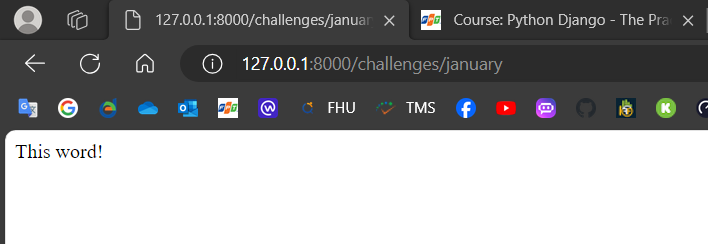
* Code ở bên trong file đó sẽ kiểu như thế này:



* Hiểu đơn giản thì đường dẫn january sẽ dẫn tới cái view được định nghĩa bên trên, bây h là cần làm sao để khi gõ cái đường dẫn là challenges/january thì cái view đó hiện ra.
* Thế là cần phải khai báo mấy cái url của app challenges với server, để server còn biết đường mà trả về, trong file urls.py của server, kiểu như thế này:



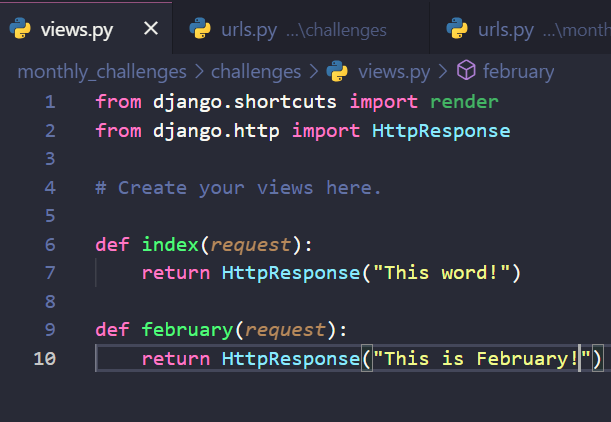
* Và giờ khi gõ đường dẫn: “[127.0.0.1:8000/challenges/january](http://127.0.0.1:8000/challenges/january)”, trình duyệt sẽ hiện ra, kiểu như thế này:



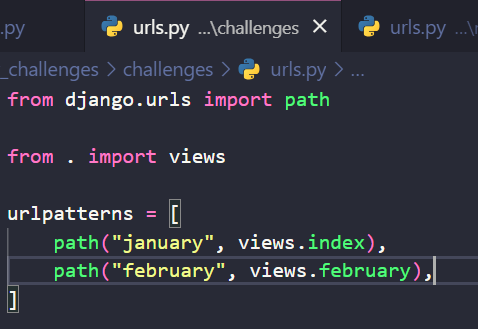
* Hmm, nếu mà nhớ không lầm thì cái hàm index trong view hình như cũng là hàm mặc định, nếu mà không có cái phần đằng sau challenges trong đường dẫn.

## 22. Adding More Views & URLs

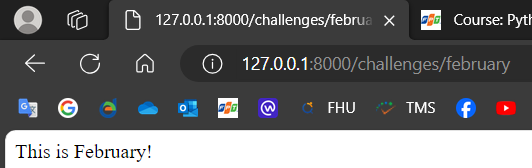
* Bên trên là tháng 1, bây h là tháng 2.
* Mong muốn là tạo một cái view để xử lý đường dẫn challenges/february
* Đầu tiên, tạo một view trong views.py của app challenges:



* Tiếp theo là thêm đường dẫn vào file urls.py của app challenges:

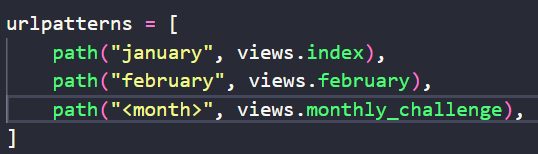


* Kết quả kiểu kiểu như thế này:

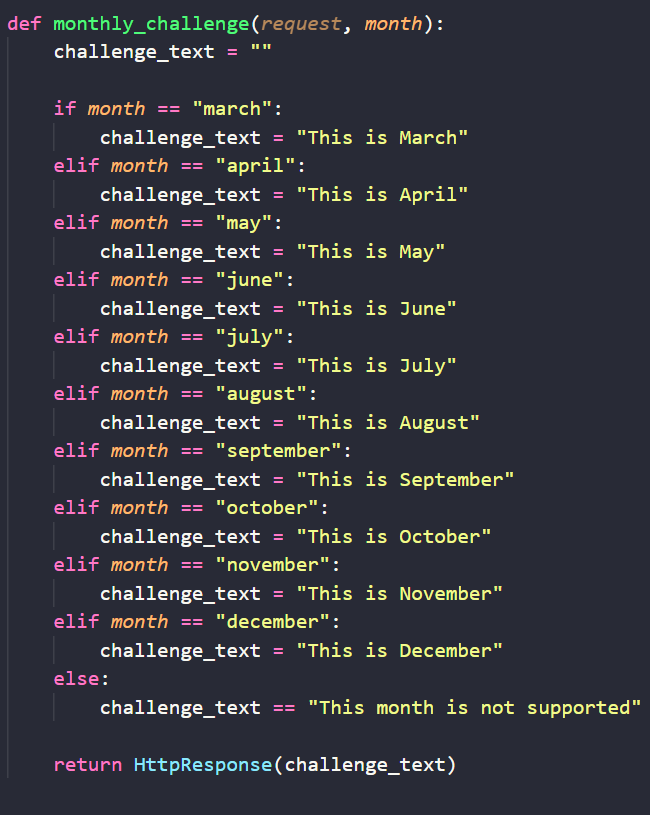
****

## 23. Dynamic Path Segments & Captured Values

* Nếu mà bây giờ đường dẫn cần xử lý cũng chỉ là dưới dạng challenges/<tháng nào đó>, thì việc viết ra một mớ hàm xử lý có vẻ là thừa thãi, và giá như có cơ chế để gộp chung việc xử lý mấy đường dẫn này lại thì tốt. Và may là có.
* Trước hết, cần tạo một kiểu mẫu cho đường dẫn, trong file urls.py của app challenges, kiểu như thế này:



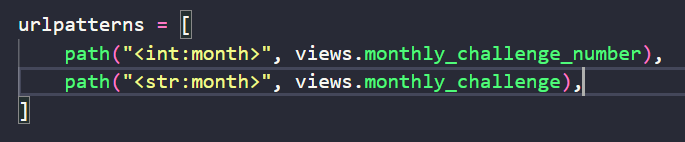
* “<month>” có thể được gọi là một placeholder, nơi mà được coi như một tham số để viết hàm xử lý đường dẫn, hàm đó kiểu như thế này:



* Tham số month được truyền vào trong hàm chính là phần được đặt trong <> ở phần khai báo đường dẫn bên trên, bây h khi nhập đường dẫn tương ứng trên trình duyệt thì sẽ nhận được những kết quả tương ứng, có thể nhìn thấy trong hàm này.
* Việc nhập tháng 1 hay tháng 2 sẽ không ảnh hưởng vì chúng đã được khai báo trước đó.

## 24. Path Converters





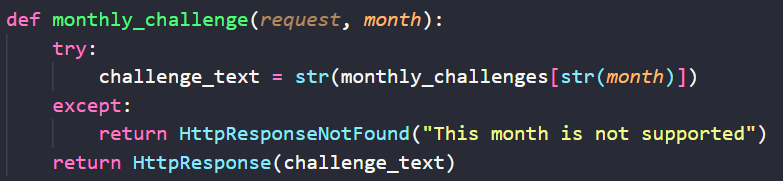
* Có thể ép kiểu tham số để truyền vào hàm để xử lý, ở đây, nếu đường dẫn chứa tham số có thể ép kiểu về int thì nó sẽ được xử lý với đường dẫn thứ nhất. Nếu đường dẫn chứa tham số có thể ép kiểu về str thì nó sẽ được xử lý với đường dẫn thứ hai.

## 25. Adding More Dynamic View Logic

* Một trong những điểm mạnh của python là nó có những cấu trúc dữ liệu mạnh mẽ được hỗ trợ sẵn. Ví dụ điển hình là dict và tuple.
* Việc xử lý đường dẫn ở bên trên có thể được thực hiện dễ dàng bằng cách sử dụng dict, không nhất thiết là phải cần các conditional statement như if else.
* Trước tiên, cần thêm một dict:



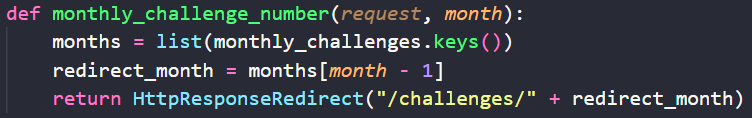
* Có thể thấy dict này chứa các key và value tương ứng với nhau, và hàm xử lý ở đây sẽ kiểu như thế này:



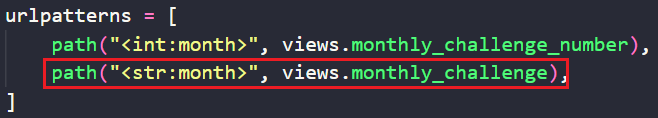
* Việc xử lý ngoại lệ với try – except ở đây là cần thiết để phòng trường hợp đường dẫn nhập vào không hợp lệ, dẫn đến việc không thể tìm thấy value tương ứng trong dict.
* Việc ép kiểu về chuỗi sử dụng str() là cần thiết nếu sau này trong dict có thể sửa đổi để sử dụng các kiểu dữ liệu nguyên thủy không phải chuỗi.
* Việc sử dụng dict như trên, chính là việc sử dụng dynamic code, nó khiến cho code linh hoạt hơn. Bên cạnh đó, code cũng dễ bảo trì hơn, hãy thử tưởng tượng đến trường hợp chúng ta có tháng thứ 13 chẳng hạn, việc đơn giản cần làm chỉ là thêm nó vào dict.

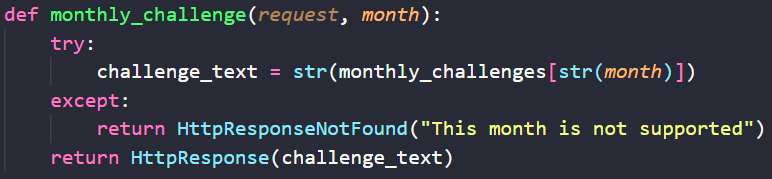
## 26. Redirects

* Redirect hay tái điều hướng là một việc cần thiết trong web, khi mà một đường dẫn có thể được xử lý dựa trên hàm xử lý đã có sẵn của một đường dẫn khác.
* Dưới đây là một ví dụ về việc tái điều hướng, với đường dẫn được tái điều hướng chính là loại đường dẫn mà tháng trong năm được nhập vào dưới dạng một số chứ không phải là một từ. Hàm xử lý của đường dẫn có dạng như sau:



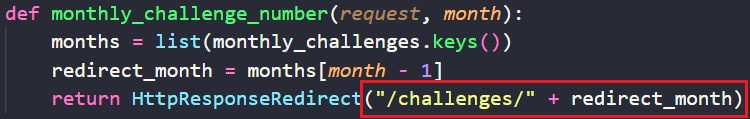
* Giải thích code:
  + Trước tiên, danh sách tháng months được lấy ra từ dict monthly\_challenges ở bên trên.
  + Redirect\_month là tháng tương ứng trong months với số mà người dùng nhập vào.
  + Sau đó, một HttpResponseRedirect được trả về, báo hiệu việc chuyển đường dẫn sang thành định dạng "/challenges/" + redirect\_month đến server, và server sẽ xử lý đường dẫn này dựa trên hàm xử lý tương ứng đã được định nghĩa cho định dạng đường dẫn này trong file urls.py:



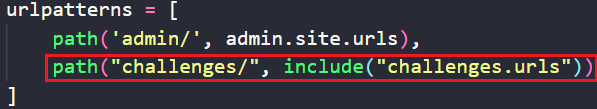


## 27. The Reverse Function & Named URLs

* Quay trở lại với hàm xử lý có liên quan đến redirect:

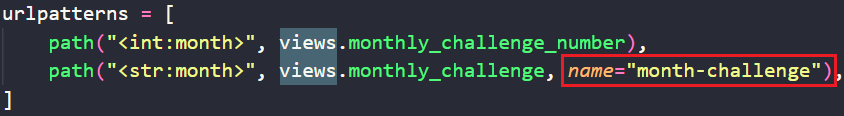


* Có thể thấy rằng, phần đường dẫn trong hàm HttpResponseRedirect là hardcode, và nếu đường dẫn:

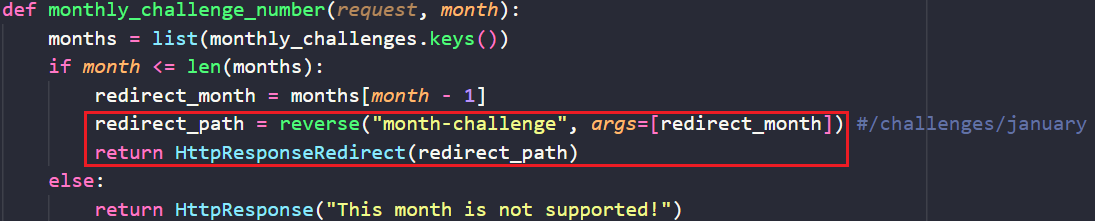


thay đổi, việc này sẽ kéo theo việc cũng phải thay đổi lại đường đẫn trong hàm HttpResponseRedirect, đây là điều cần tránh. Và để làm việc đó, Django cung cấp hàm reverse().

* Để có thể sử dụng được hàm này, trước tiên cần đặt một “name” cho đường dẫn cần redirect đến:



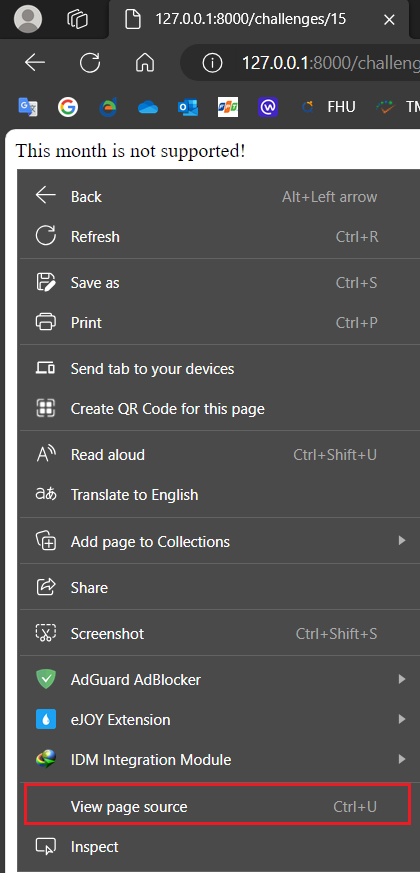
* Và sau đó là chỉnh sửa hàm xử lý của đường dẫn cần phải redirect:



* Hàm reverse ở đây nhận vào 2 tham số, tham số thứ nhất là tên của đường dẫn, cho biết xem đường dẫn được redirect đến là kiểu đường dẫn nào trong app tương ứng, tham số thứ hai chính là nội dung của đường dẫn, ở đây chính là tháng tương ứng với số trên đường dẫn ban đầu.

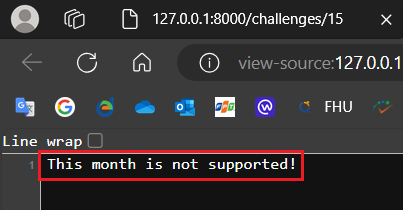
## 28. Returning HTML

* Một điều hiển nhiên trong thực tế là các trang web không chỉ trả về mấy đoạn văn bản đơn giản, mà đó là những file HTML có giao diện. Bây giờ hãy thử làm một chút ví dụ. Còn việc làm nghiêm túc sẽ được thực hiện ở Section sau.
* Khi chưa thực hiện chỉnh sửa gì, hãy chọn:

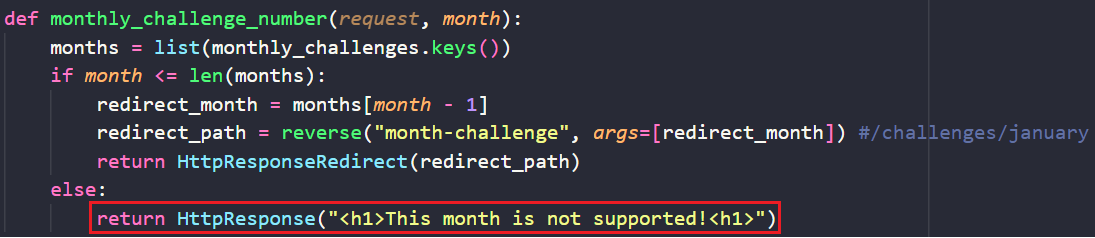


trên cửa sổ trình duyệt

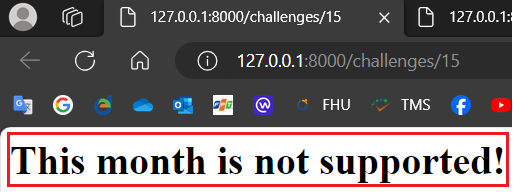
* Kết quả sẽ kiểu như thế này:



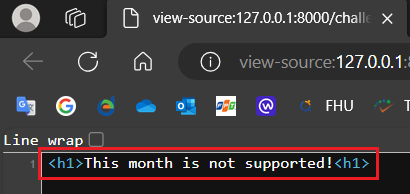
* Hãy chỉnh sửa source code:



* Và kết quả sẽ kiểu như thế này:

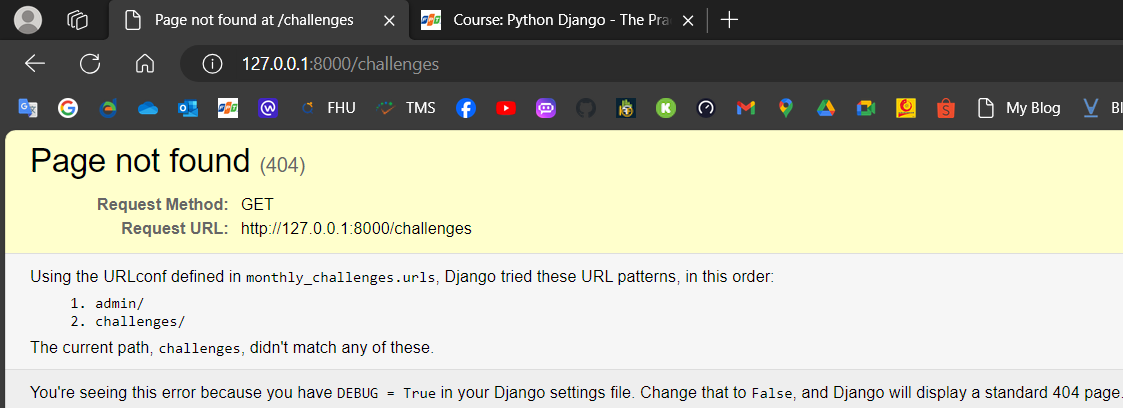


* Và khi view page source:

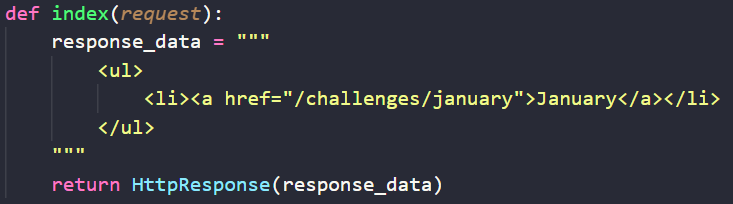


## 29. Practicing URLs, Views & Dynamic View Logic

* Đến bây giờ thì vẫn chưa có view cho đường dẫn mặc định, kiểu như nhập vào là “/challenges/”:



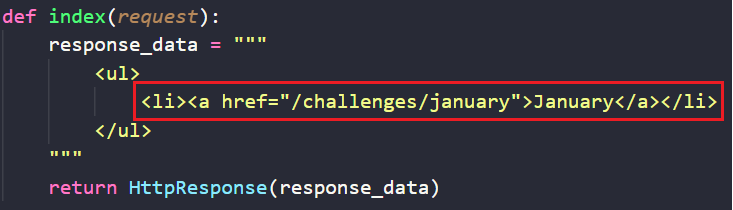
* Theo concept đơn giản thì khi gõ đường dẫn này thì sẽ hiện ra danh sách các tháng. So, let’s go.
* Hàm xử lý đường dẫn:



* Và link nó vào đường dẫn trong file urls.py:

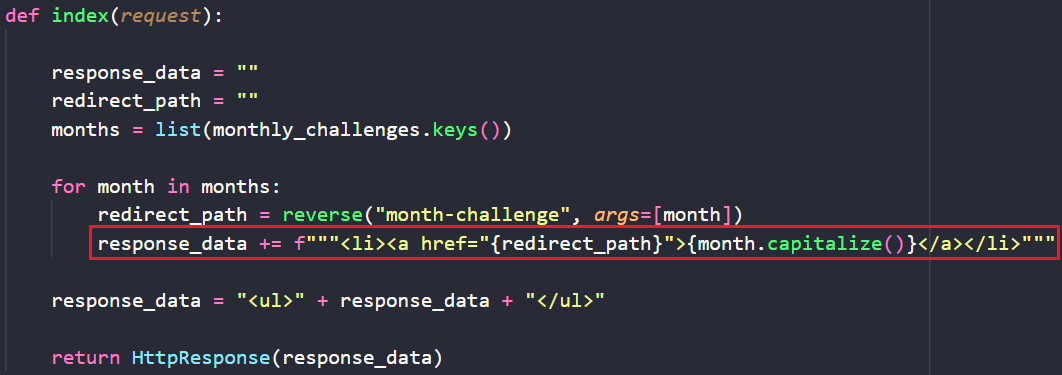


* Nhưng mà phần code:

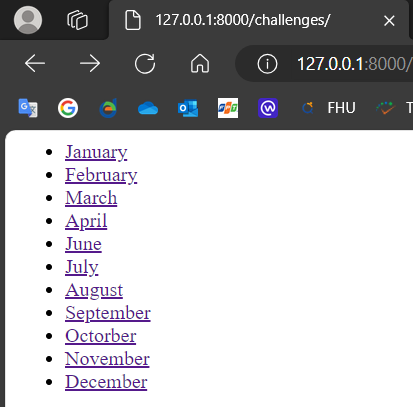


không có make sense, tại vì nó hardcode là một, và thứ hai la sẽ phải viết code cho mỗi tháng.

* Như vậy, cần phải viết code sao cho xử lý được 2 vấn đề này:



* Ký tự “f” phía trước chuỗi biến chuỗi đó thành một f-string, là một trong những cách để có thể đẩy giá trị của biến vào trong một chuỗi.
* Tên hàm là “index”, đây là một kiểu tiêu chuẩn trong web, khi tên “index” hay “index.html” là tên của những hàm hay file mặc định. Ở đây hàm index là hàm xử lý đường dẫn mặc định.
* Kết quả kiểu như thế này:



## 30. Summary

## 31. Useful Resources & Links

# Section 4: Templates & Static Files

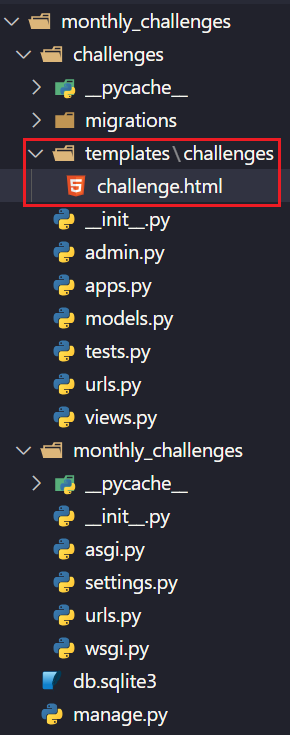
## 32. Module Introduction

Module Content:

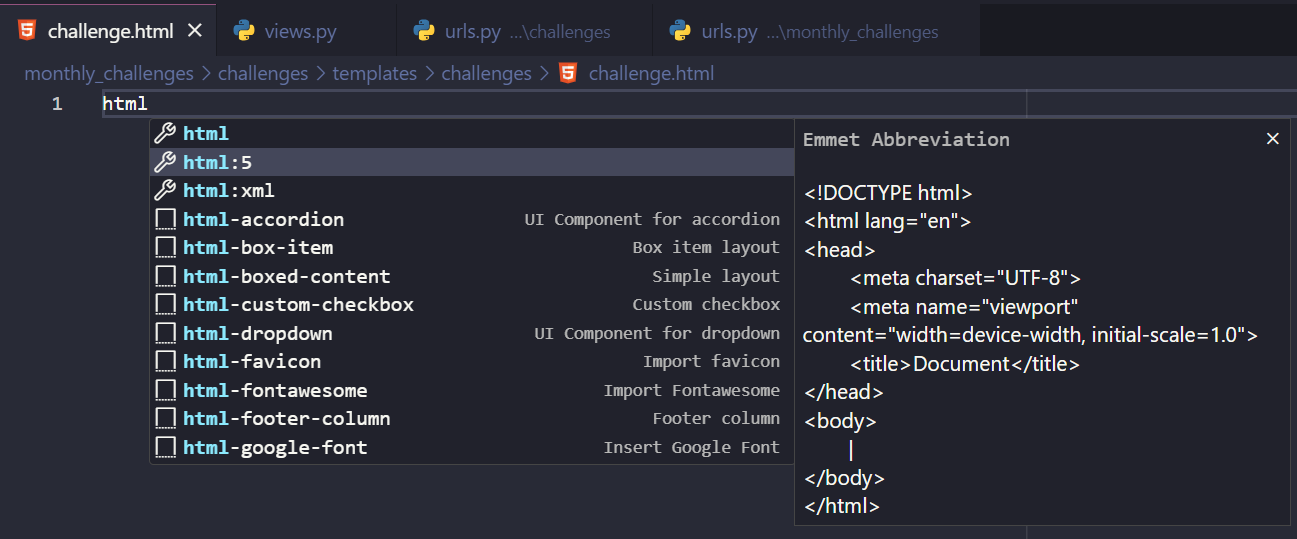
* Understanding “Templates”
* Django Template Language Features
* Working with Static Files (CSS, JavaScript, Images)

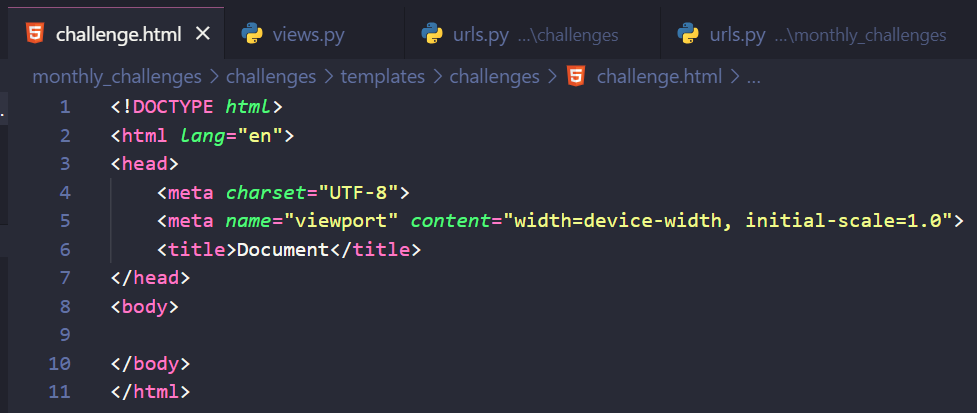
## 33. Adding & Registering Templates

* Templates có thể hiểu đơn giản là thứ nắm giữ những file HTML để có thể tạo ra những dynamic content. Như vậy, có nghĩa là những phần code HTML ở các code trong view phía trên sẽ được tạo ra ở một nơi khác.
* Trước tiên là vị trí đặt file HTML:

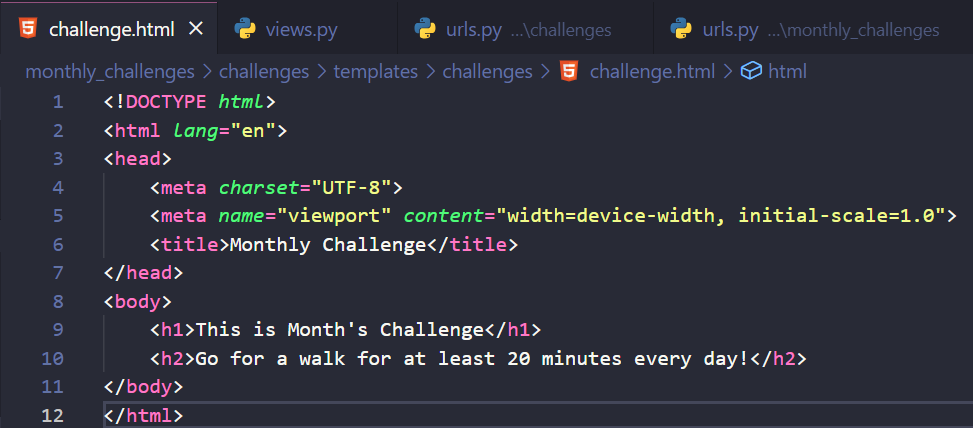


* Các file HTML của app (ở đây là challenges) sẽ được đặt trong một thư mục con có tên của app đó, thư mục này được đặt vào trong một thư mục tên là templates và thư mục template đó lại được đặt vào trong chính thư mục của app.
* Việc đặt tên thư mục như phía trên là bắt buộc, việc đó được coi là một convention của django và phải được tuân theo để đảm bảo việc hoạt động chính xác.
* Nội dung của file HTML:

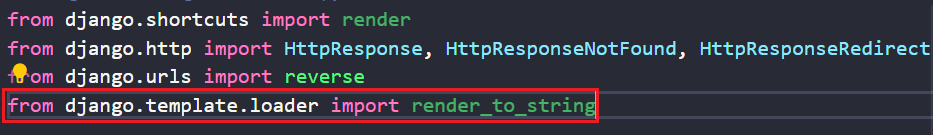




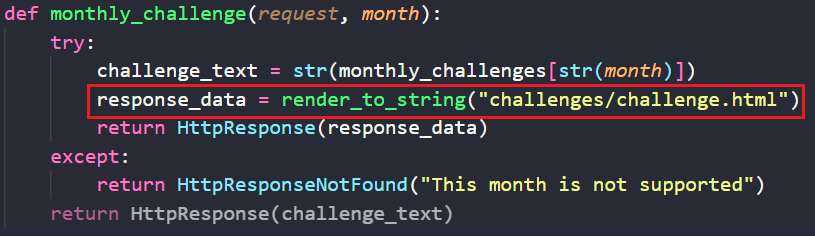
* Hãy tạo một ví dụ đơn giản:



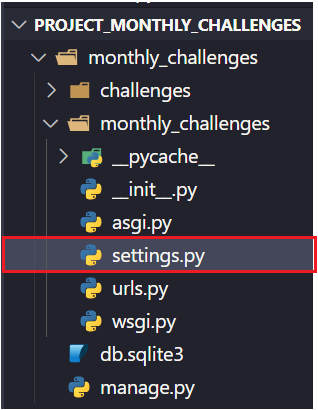
* Việc cần làm bây giờ là viết code trong hàm xử lý để có thể trả về file HTML này.
* Có thể hiểu đơn giản là có thể đẩy các HTML code vào trong hàm HttpResponse. Để có thể chuyển đổi file HTML thành string, cần sử dụng hàm render\_to\_string:



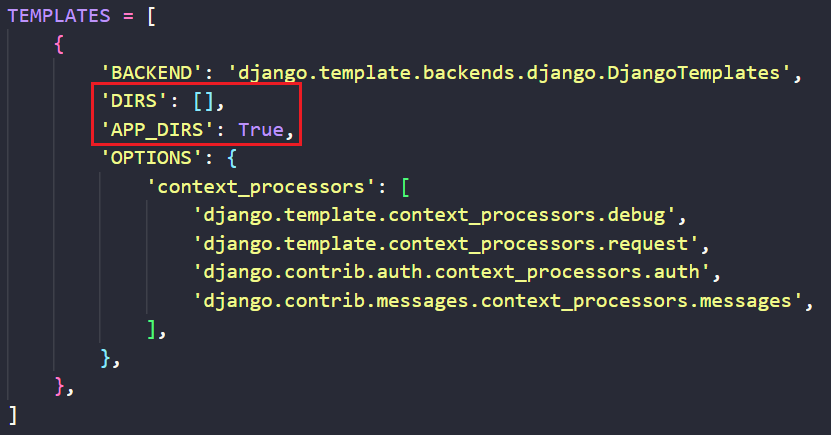
* Hàm xử lý được viết lại như sau:



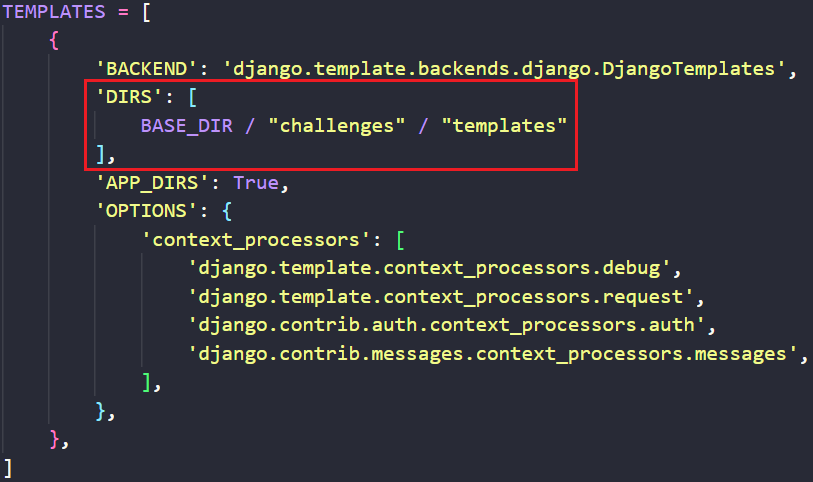
* Bây giờ, cần thông báo cho server biết xem cần tìm file template ở đâu, việc này được cấu hình ở trong file settings.py trong thư mục chính của project:



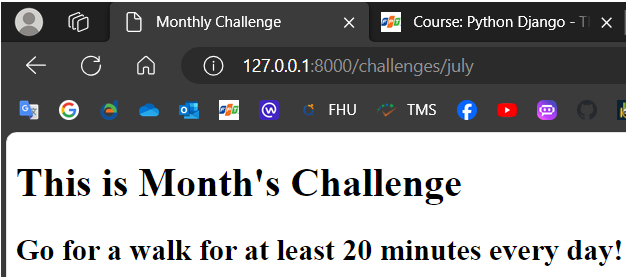
* Trong file này, tìm đến phần cấu hình template:



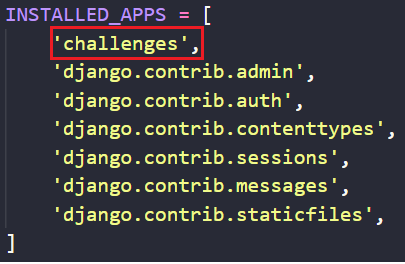
* Và cần phải cấu hình lại thuộc tính “DIRS” như sau:



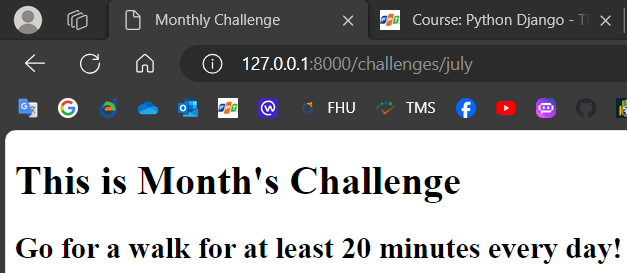
* Và kết quả bây giờ sẽ kiểu như thế này:



* Tuy nhiên, có một vấn đề. Việc cấu hình thuộc tính “DIRS” như phía trên là dành cho global templates, hay đó là template của cả project. Điều này là bất ổn, bởi trong một project có thể có nhiều app.
* Do đó, cách tốt nhất là khai báo cái app đó tới hệ thống, và hệ thống sẽ tự tìm template của app đó khi cần.
* Trước tiên, hãy xóa phần cấu hình thuộc tính “DIRS” ở phía trên. Và vẫn trong file settings.py, tìm đến phần cấu hình INSTALLED\_APPS, và cấu hình thêm như sau:

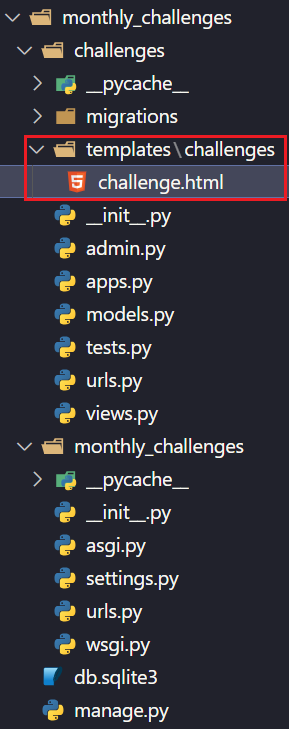


* Và kết quả bây giờ sẽ kiểu như này:

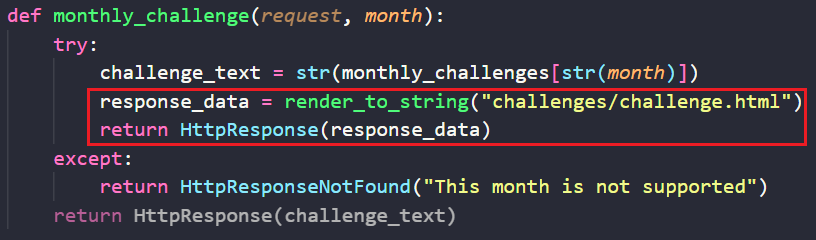


## 34. Rendering Templates

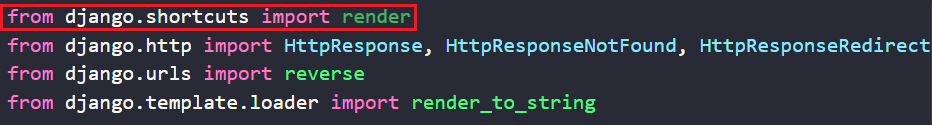
* Quay lại một chút về vị trí của file HTML:



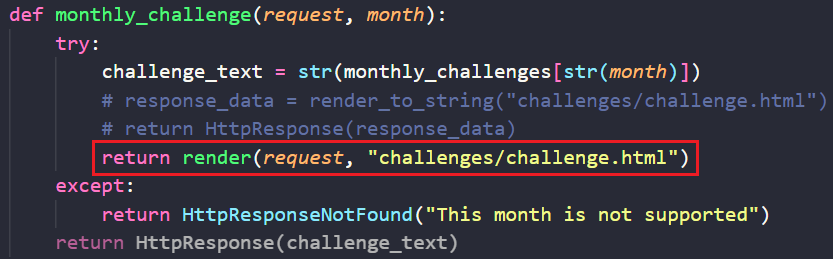
* Tại sao lại không đặt file HTML ra luôn bên ngoài?. Điều này là hoàn toàn có thể, và với việc chỉnh sửa lại đường dẫn trong hàm xử lý, trong trường hợp này, mọi thứ vẫn hoạt động bình thường.
* Tuy nhiên, trong thực tế, có thể trong project sẽ có nhiều app. Django chỉ đơn giản là lấy mọi thứ trong thư mục templates của mỗi app và ném chúng vào chung một chỗ. Và nếu như có một app khác cũng có một template có tên là challenge.html, thì khi cần, server sẽ không thể biết được là nó cần sử dụng template nào. Do đó, việc đặt các file template vào trong thư mục mang tên của app sẽ khiến cho server có thể tìm được đúng đến template mà nó cần generate.
* Quay trở lại với hàm xử lý:



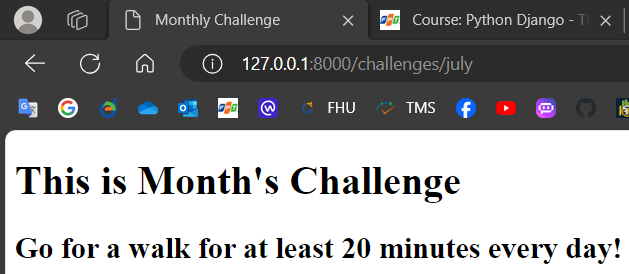
* Hàm này có thể được viết lại ngắn hơn với render method:



* Hàm xử lý được viết lại như sau:

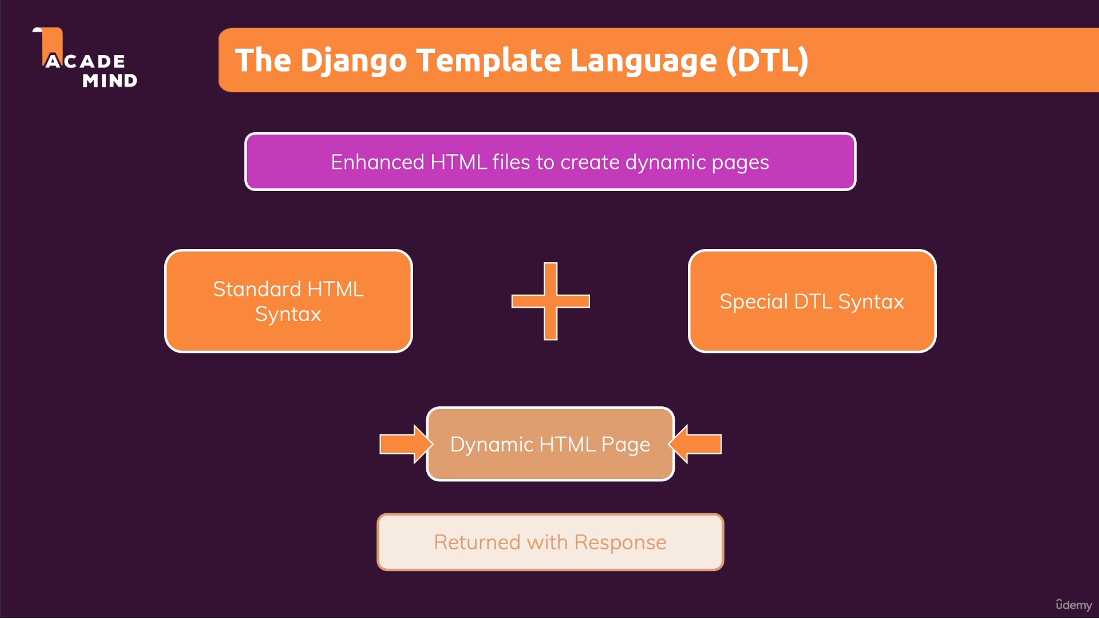


* Và kết quả sẽ vẫn tương tự:

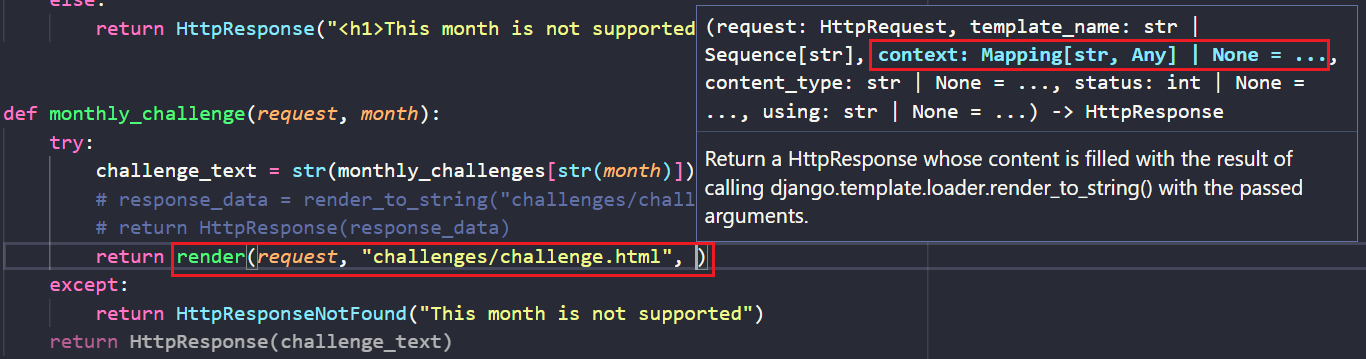


## 35. Template Language & Variable Interpolation

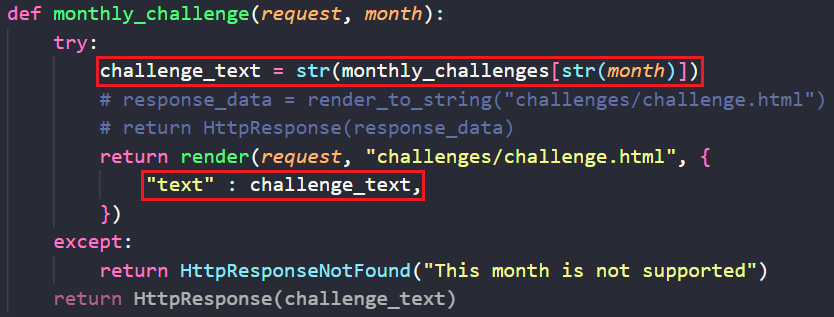
* Trong thực tế, html được trả về cho người dùng không thể nào được tạo tĩnh ở trong template, html được trả về phải dựa trên thứ mà người dùng yêu cầu, và thường phải được viết để có thể đẩy dữ liệu tùy chỉnh vào.



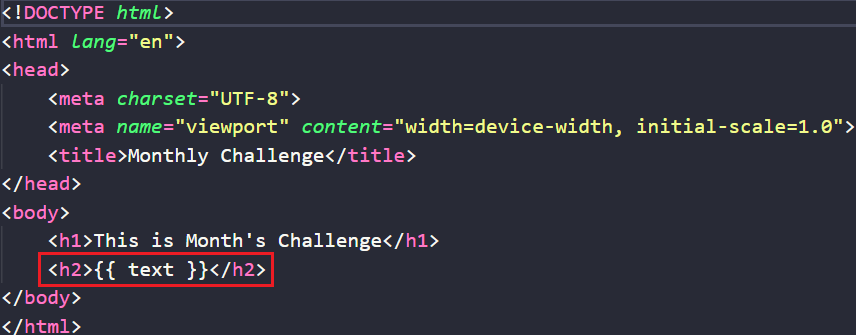
* Trong django, bằng cách sử dụng html tiêu chuẩn kết hợp với Django Template Language (DTL), có thể tạp ra các html động để trả về cho người dùng. Dưới đây là một ví dụ.
* Trong hàm render, tham số được truyền vào thứ 3 được gọi là context của template:



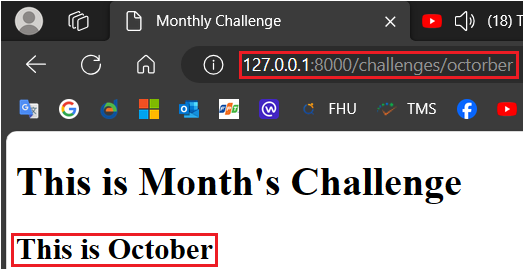
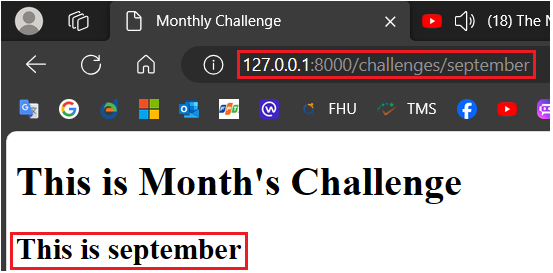
* Tham số này được truyền vào dưới dạng một dict, kiểu như:



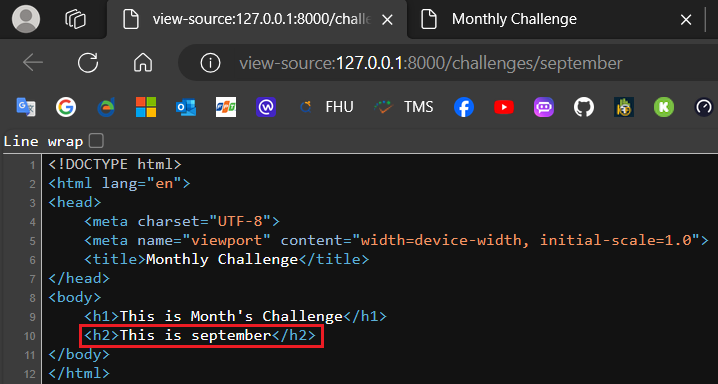
* Và trong file html của template, “{{}}” là một cú pháp đặc biệt được define bởi django:



* Và kết quả kiểu như thế này:

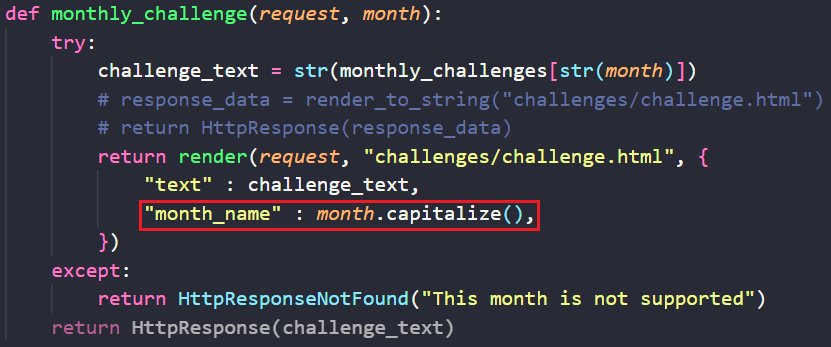


* Và khi view page source, có thể thấy nội dung trong file html đã được thay thế:

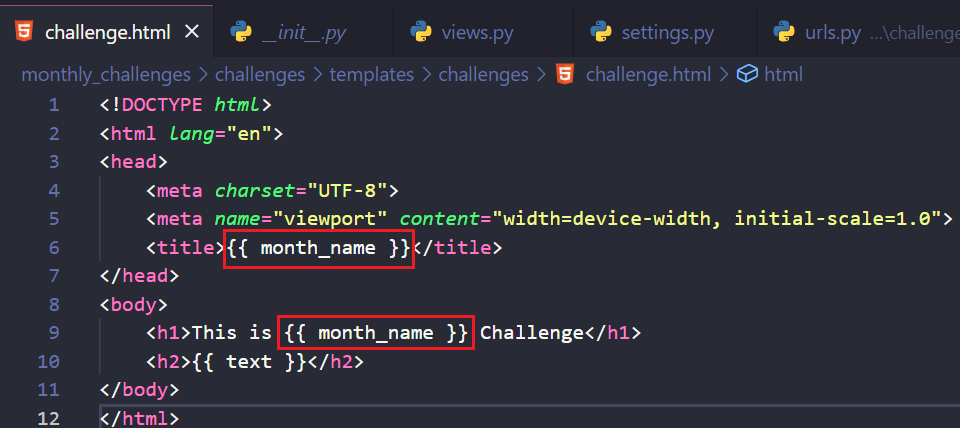


## 36. Exercise Solution

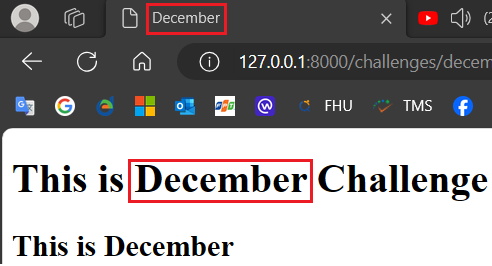
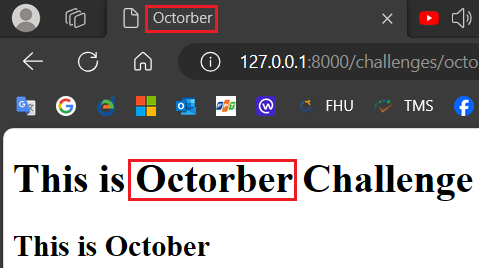
* Trong bài tập này, mục tiêu là chỉnh sửa các tag <title> và <h1> trong html, từ việc chúng mang giá trị tĩnh thành giá trị động.
* Trước tiên, cần tạo context cho chúng:



* Sau đó là chỉnh sửa file html:

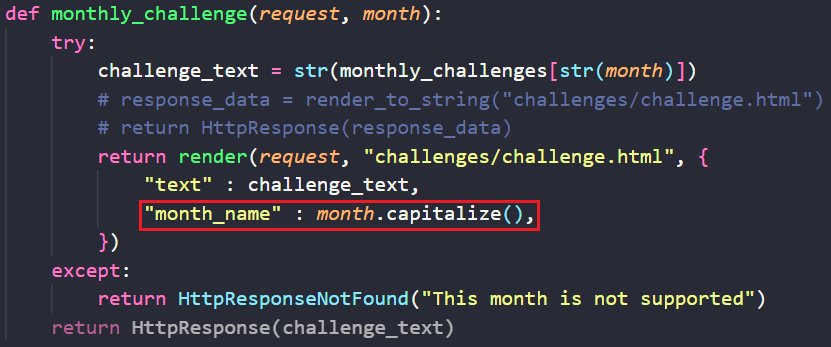


* Và kết quả sẽ kiểu như thế này:

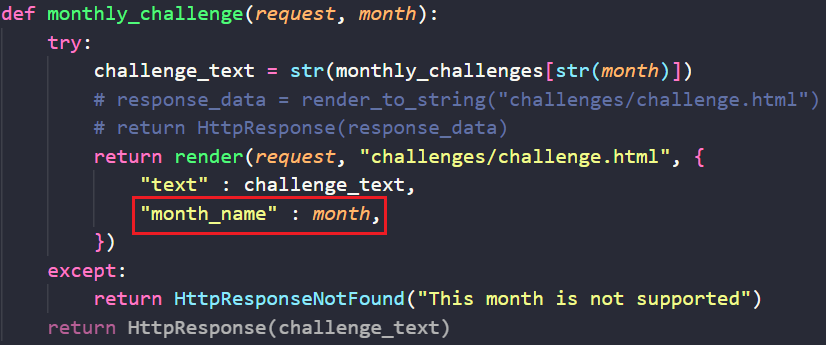


## 37. Filters

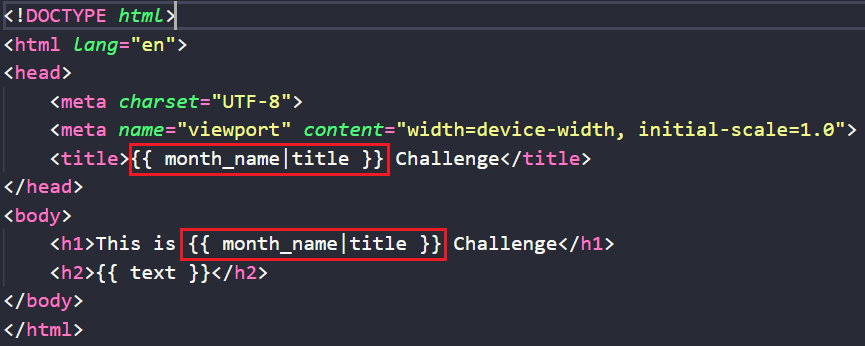
* Quay lại với một hình ảnh ở bài 36:



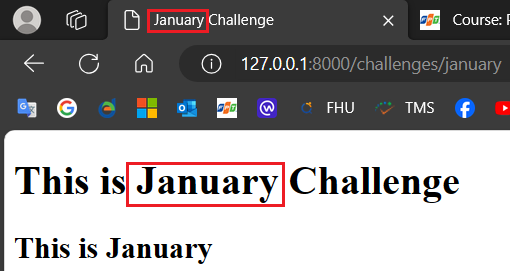
* Trong thực tế, việc định dạng văn bản được hiển thị trong html thường phải được xử lý ở phía file html, chứ không phải được xử lý luôn ở hàm xử lý đường dẫn:



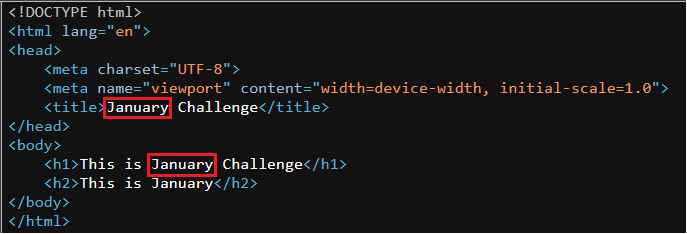
* Tuy nhiên, DTL (Django Template Language) không hỗ trợ việc chạy các logic python trên file html, tức là không thể gọi hàm capitalized() ở trong file html. Tuy nhiên, điều quan trọng là nó được xây dựng với các build-in function, một trong những build-in function quan trọng là ***tags and filter***. Ở đây ta sẽ tìm hiểu về ***filter*** trước.
* Có thể hiểu đơn giản ***filter*** là phương tiện để ***áp dụng*** một số ***xử lý*** vào một đối tượng. Trong trường hợp ở đây, ta đang muốn viết hoa chữ cái đầu của tên tháng, và đơn giản chỉ cần làm như thế này:



* Và kết quả sẽ là:



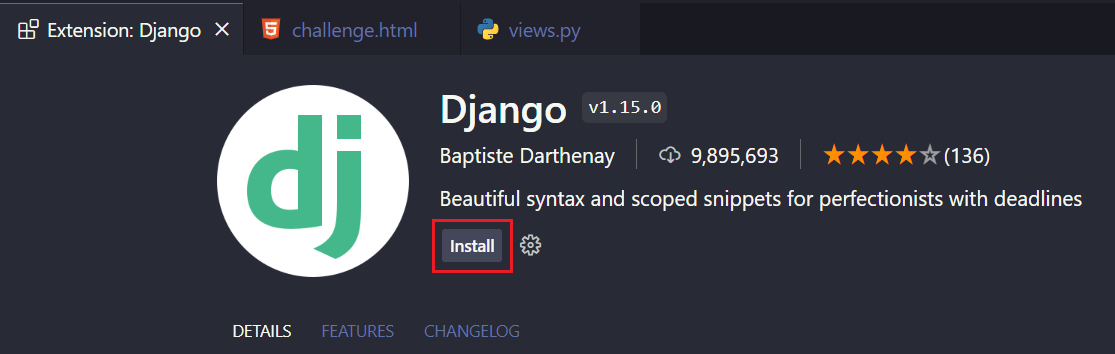
* Khi view page source, có thể thấy sự thay thế đã diễn ra:



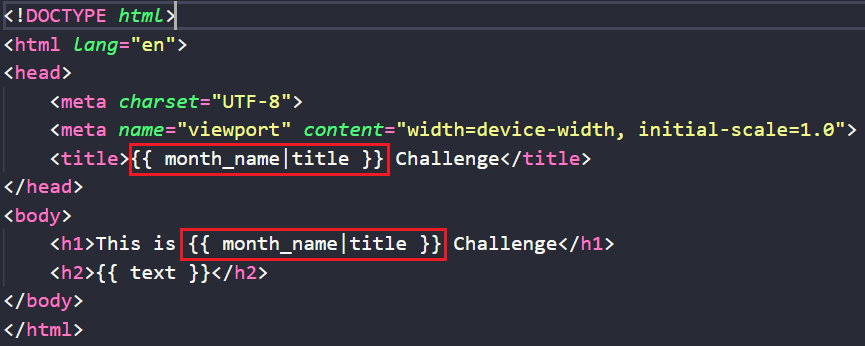
* ***tags và filter là một phần rất quan trọng trong django và dev nên đọc thêm về chúng, một trong những nguồn quan trọng là trên web chính thức của django***.

## 38. The Django Visual Studio Code Extension

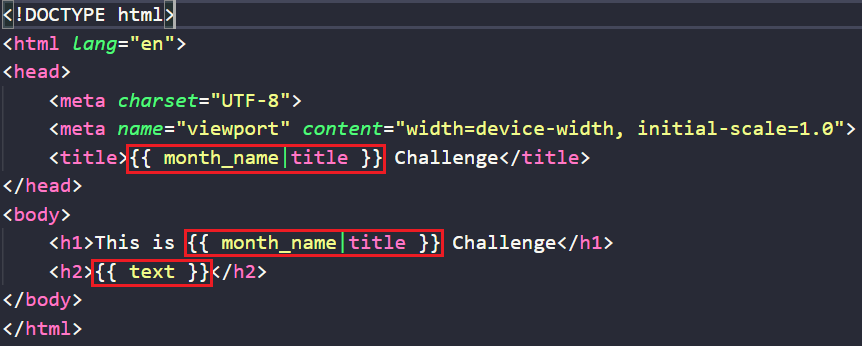
* Khi tạo các template html file, có thể thấy rằng, đôi khi, dù có viết sai định dạng thì cũng không có một cảnh báo nào hiện ra để cảnh báo dev.
* Và trong Visual Studio Code, có một extension để giúp dev trong chuyện này:



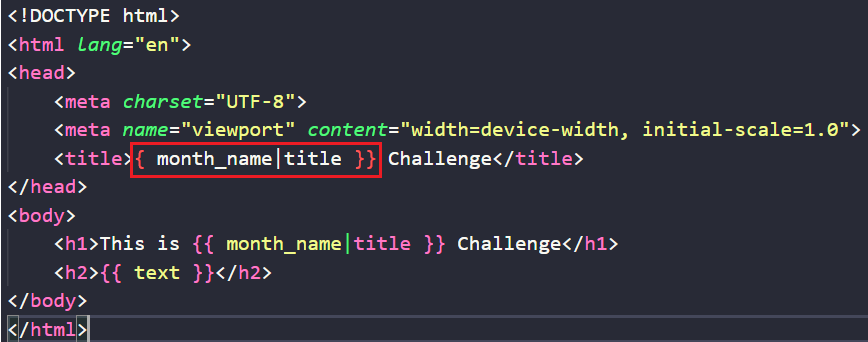
* Và sau khi cài đặt, đơn giản nhất, file template html file sẽ chuyển đổi cách hiển thị từ:



sang như thế này:

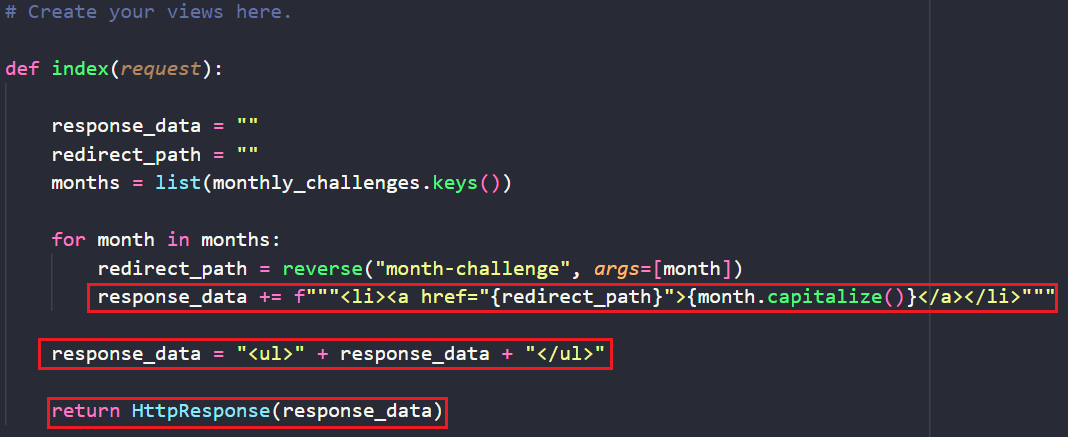


* Và khi gõ sai định dạng, mọi thứ sẽ kiểu như thế này, không có màu và mấy cái ngoặc sẽ có màu đỏ lòm:

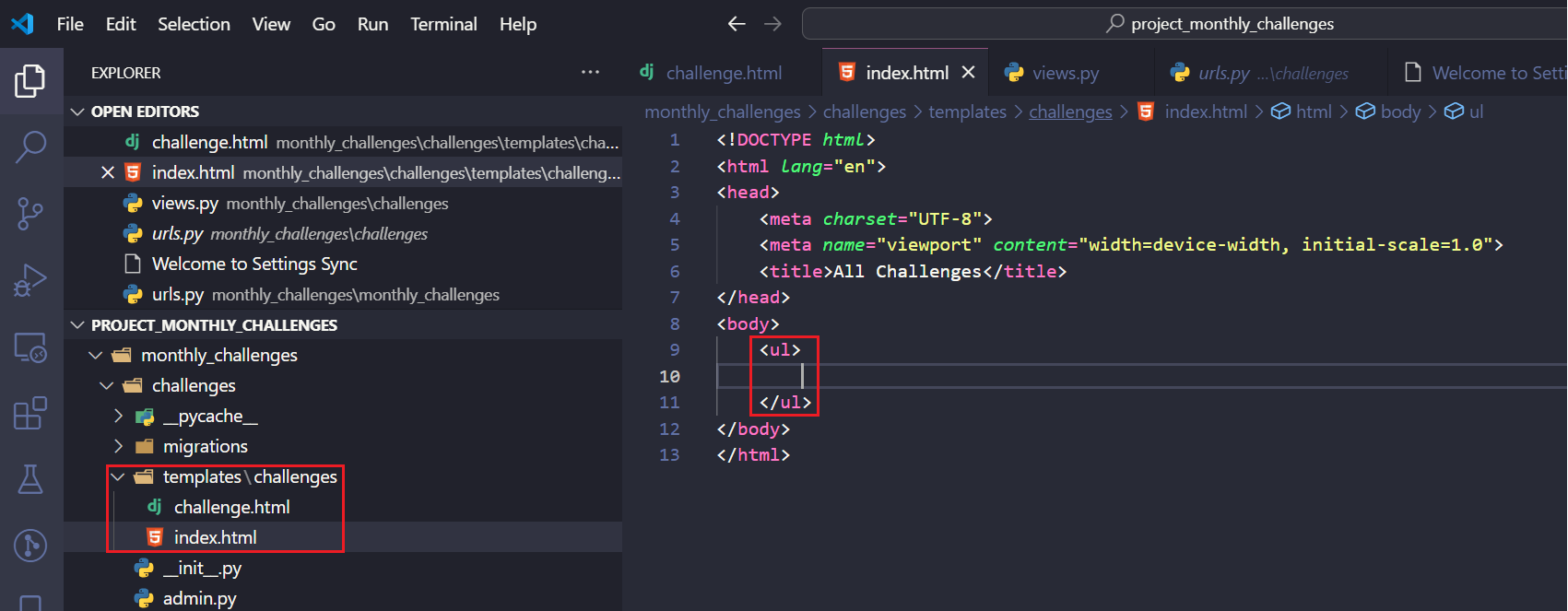


## 39. Tags & the “for” tag

* Bên cạnh filter đã được giới thiệu trước đó, “tag” cũng là một trong những chức năng quan trong nhất của Django Template Language (DTL).
* Tài liệu về “tag” có thể đọc được kĩ nhất ở trang chủ của django.
* Ở đây, ta chỉ tìm hiểu về tag “for”, là tag cho phép duyệt qua các phần tử của một danh sách trong python (list, tuple, dict,…).
* Ở các phần trước, đường dẫn mặc định “[127.0.0.1:8000/challenges/](http://127.0.0.1:8000/challenges/)” chỉ được xử lý bằng cách tạo và đẩy thẳng nội dung html vào thẳng hàm xử lý:



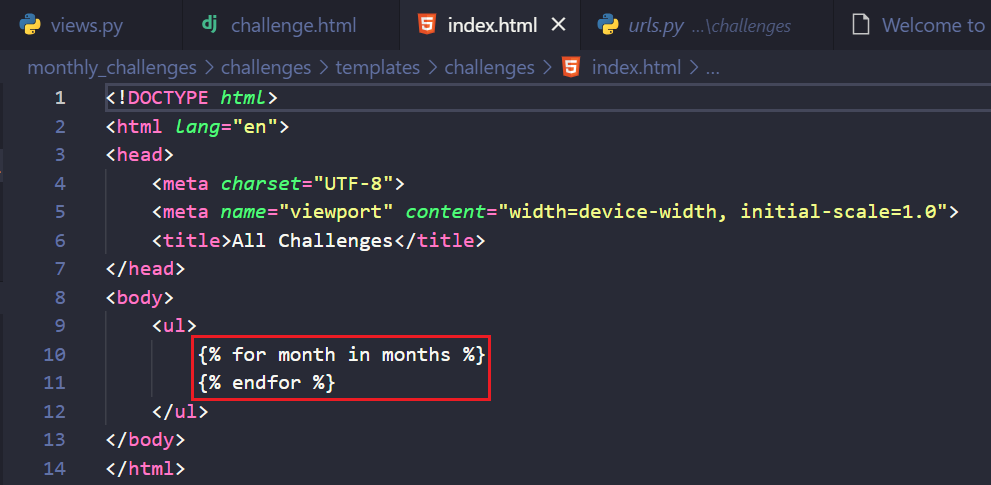
* Với việc xử dụng tag “for”, mục tiêu ở đây là việc xử lý để hiển thị danh sách tháng sẽ diễn ra ở trong file html. Điều này nghe có vẻ đúng vì việc xử lý này khá đơn giản và có thể thực hiện ở phía front-end.
* Trước tiên, cần tạo file index.html và khung sườn của nó trong thư mục template tương ứng, và tạo sẵn luôn một danh sách với HTML tag <ul> trong đó:



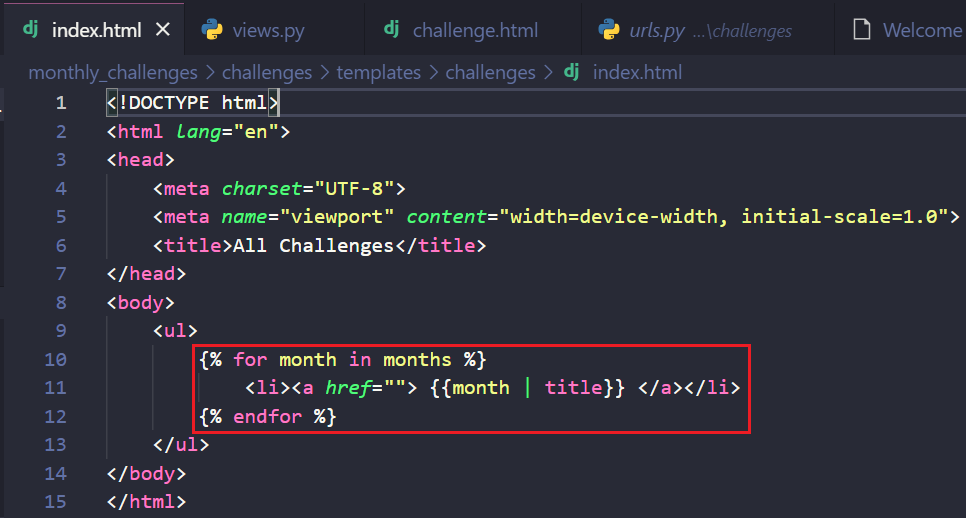
* Mục tiêu ở đây là làm sao để các phần tử của danh sách (<li> - list item) có thể được tạo ra một cách tự động bằng cách sử dụng tag “for”.
* Trước tiên, ta cần gửi danh sách các tháng ra ngoài front-end bằng cách sử dụng context trong hàm render():



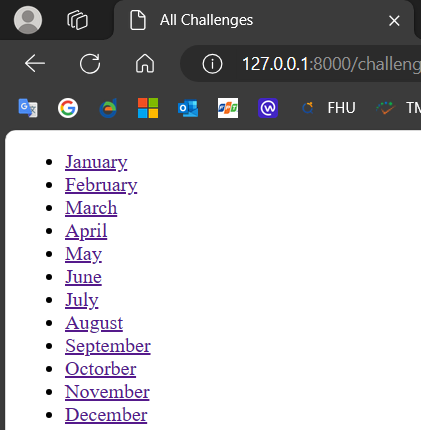
* Đoạn mã được dùng để làm điều đó có thể được gọi là một block, nơi mà một đoạn mã được bọc lại, ở đây chính là “tag”:



* Mỗi tag trong DTL có cách đóng và mở riêng. Điều này là khác biệt với python thông thường, khi mà tab được xác định xem block đã kết thúc hay chưa.
* Các phần tử trong list (<li> - list item) sẽ được tạo ra bằng đoạn mã sau:



* Một biến có kiểu dữ liệu là kiểu dữ liệu phức tạp (complex data type) như months sẽ không có ý nghĩa khi nằm ngoài một block.
* Và bây giờ, khi truy cập vào đường dẫn mặc định (đã đề cập ở trên), kết quả sẽ kiểu như thế này:



* Dĩ nhiên là ấn vào link thì sẽ không ra gì cả, vì đã tạo đâu.

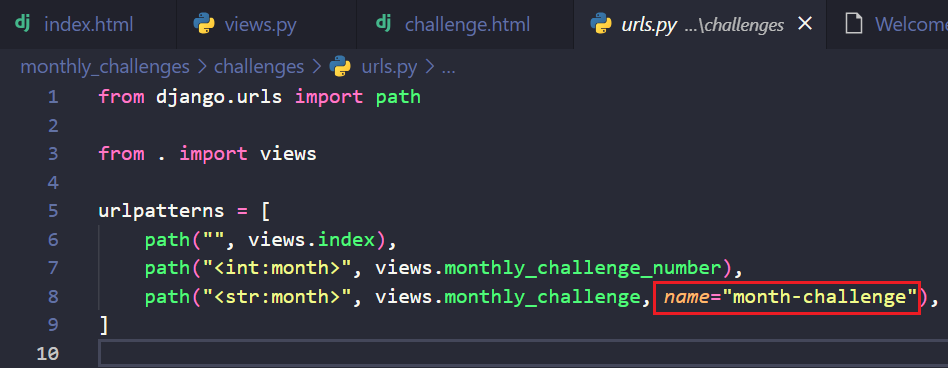
## 40. The URL Tag for Dynamic URLs

* Trong các bài trước, dynamic url được tạo ra bằng cách sử dụng hàm reverse(), nhưng hàm này không thể chạy trong một block code ở phía front-end.
* Tuy nhiên, DTL có hỗ trợ tag “url” ở phía front-end, tag này có tác dụng gần giống như hàm reverse() ở phía front-end.
* Cú pháp của tag “url”:



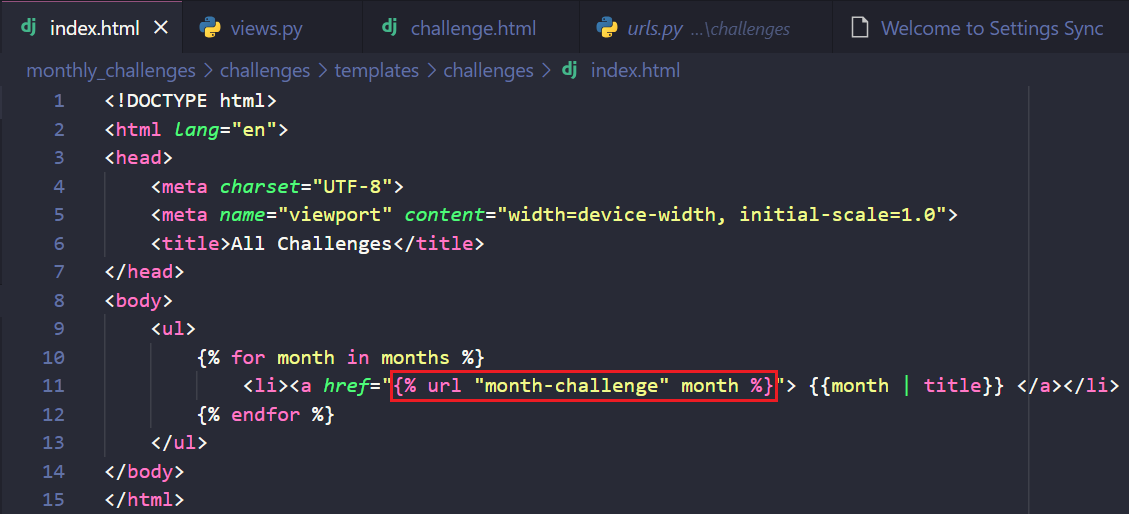


Với ‘some-url-name” chính là tên của url pattern:



Và arg1, arg2 là các thành phần của đường dẫn.

* Và trong ví dụ này, nó sẽ kiểu như thế này:



* Và mọi thứ hoạt động kiểu:

