**https://netbasal.com/understanding-viewchildren-contentchildren-and-querylist-in-angular-896b0c689f6e**

**@ViewChildren —**

*Returns the specified elements or directives from the view DOM as QueryList*

For example, let’s create a simple alert component.

Now let’s use this component multiple times in our app component and use the @ViewChildren decorator.

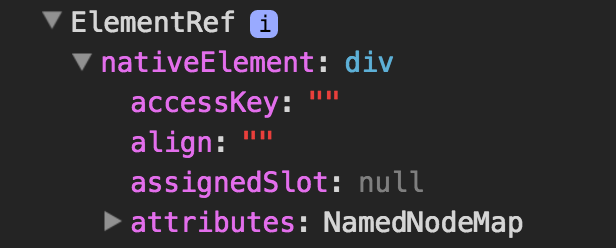
We can use the @ViewChildren decorator to grab elements from the host view.

The @ViewChildren decorator supports directive or component type as parameter, or the name of a template variable.

When the parameter is a component/directive the return value will be the component/directive instance.



When the parameter is the name of a template variable, the return value will be a reference to the native element.

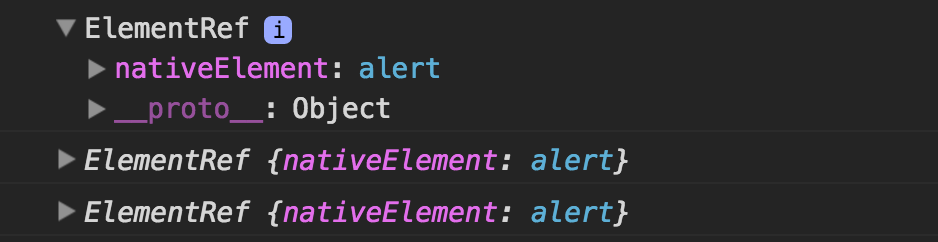


**Note:** In this case, it will make more sense to use the @ViewChild decorator because this is only a single element.

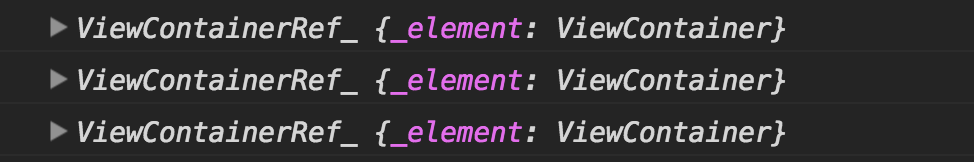
**The read parameter —**

As we said by default, the ViewChildren decorator will return the component instance, but you can ask for other tokens:

* The native DOM element —



* [ViewContainerRef](https://www.google.co.il/url?sa=t&rct=j&q=&esrc=s&source=web&cd=14&cad=rja&uact=8&ved=0ahUKEwjQyqX67NbSAhVHuhQKHeH2Dt04ChAWCCkwAw&url=https%3A%2F%2Fnetbasal.com%2Fdynamically-creating-components-with-angular-a7346f4a982d&usg=AFQjCNF-3pJQ-kDTlmQaw8NC3FPVyftJLA&sig2=Vc10x9lR4tk4uorupwCnfQ&bvm=bv.149397726,d.d24) — You need this token when you need to create templates or components dynamically



**QueryList —**

The return type of ViewChildren is QueryList. QueryList is just a fancy name for an object that stores a list of items. What is special about this object is when the state of the application changes Angular will automatically update the object items for you.

QueryList implements an iterable interface, therefore, it can be used in Angular templates with the ngFor directive. ( you can read more about this topic [here](https://netbasal.com/angular-2-ngfor-array-with-unique-values-6b15478d6484#.letjx3svz) )

**QueryList API —**

Getters —

* first — get the first item
* last — get the last item
* length — get the items length

Methods —

[map](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Array/map)(), [filter](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Array/filter)() , [find](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Array/find)(), [reduce](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Array/reduce)(), [forEach](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Array/forEach" \t "_blank)(), [some](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Array/some)().

* toArray() — returns the items as javascript array
* changes() — Changes can be observed by subscribing to the changes Observable. Any time a child element is added, removed, or moved, the query list will be updated, and the changes observable of the query list will emit a new value.

**Remember —**

The QueryList is initialized only before the ngAfterViewInit lifecycle hook, therefore, is available only **from** this point.

**ViewChildren vs ContentChildren —**

ViewChildren **don’t** include elements that exist within the ng-content tag.

ContentChildren includes **only** elements that exists within the ng-contenttag.



**@ContentChildren —**

*Returns the specified elements or directives from the content DOM as QueryList*

**Remember —**

The QueryList is initialized only before the ngAfterContentInit lifecycle hook, therefore, is available only from this point.

Top of Form

Bottom of Form

**[](https://www.freecodecamp.org/news)**

* [Learn](https://learn.freecodecamp.org/)
* [Forum](https://www.freecodecamp.org/forum/)
* [News](https://www.freecodecamp.org/news/)

29 OCTOBER 2018/[#TECH](https://www.freecodecamp.org/news/tag/tech/)

# Everything you need to know about ng-template, ng-content, ng-container, and \*ngTemplateOutlet in Angular



#### [Prateek Mishra](https://www.freecodecamp.org/news/author/patrick1729/)

Read [more posts](https://www.freecodecamp.org/news/author/patrick1729/) by this author.

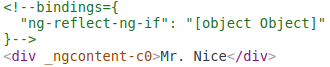
 Tweet this to your followers

Email this to a friend

Share this with your friends



It was one of those days when I was busy working on new features for my office project. All a sudden, something caught my attention:

Final rendered DOM in Angular

While inspecting the DOM I saw the ngcontent being applied on elements by Angular. Hmm… if they contain the elements in the final DOM, then what is the use of <ng-container>? At that time I got confused between <ng-container> and <ng-content>.

In the quest to know the answers to my questions I discovered the concept of <ng-template>. To my surprise, there was also \*ngTemplateOutlet. I started my journey seeking clarity about two concepts but now I had four of them, sounding nearly the same!

Have you ever been to this situation? If yes, then you are in the right place. So without further ado let’s take them one by one.

### 1. <ng-template>

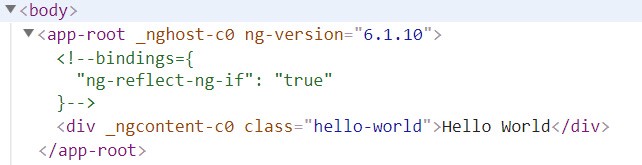
As the name suggests the <ng-template> is a template element that Angular uses with structural directives (\*ngIf, \*ngFor, [ngSwitch] and custom directives).

**These template elements only work in the presence of structural directives**. Angular wraps the host element (to which the directive is applied) inside <ng-template> and consumes the <ng-template> in the finished DOM by replacing it with diagnostic comments.

Consider a simple example of \*ngIf:

**Example 1**- Angular process of interpreting structural directives

Shown above is the Angular interpretation of \*ngIf. Angular puts the host element to which the directive is applied within <ng-template> and keeps the host as it is. The final DOM is similar to what we have seen at the beginning of this article:

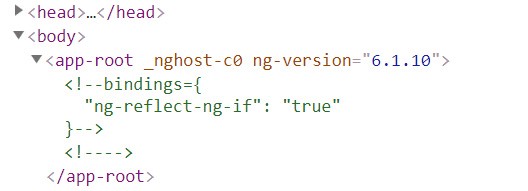
**Example 1**- Final rendered DOM

#### Usage:

We have seen how Angular uses <ng-template> but what if we want to use it? As these elements work only with a structural directive, we can write as:

**Example 2**- Using <ng-template>

Here home is a boolean property of the component set to true value. The output of the above code in DOM:

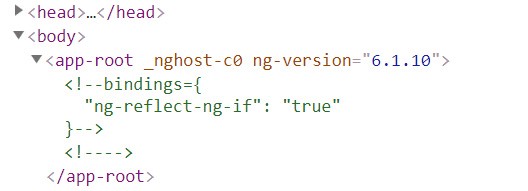
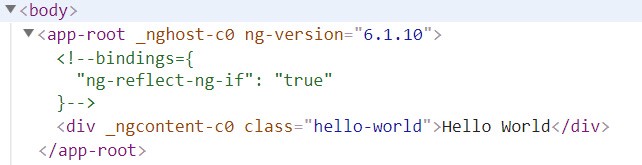
**Example 2**- Final rendered DOM

Nothing got rendered! :(

But why can’t we see our message even after using <ng-template> correctly with a structural directive?

This was the expected result. As we have already discussed, Angular replaces the <ng-template> with diagnostic comments. No doubt the above code would not generate any error, as Angular is perfectly fine with your use case. You would never get to know what exactly happened behind the scenes.

Let’s compare the above two DOMs that were rendered by Angular:

**Example 1**vs **Example 2**

If you watch closely, there is one **extra comment tag** in the final DOM of **Example 2**. The code that Angular interpreted was:

Angular interpretation process for **Example 2**

Angular wrapped up your host <ng-template> within another <ng-template> and converted not only the outer <ng-template> to diagnostic comments but also the inner one! This is why you could not see any of your message.

To get rid of this there are two ways to get your desired result:

Correct usage of <ng-template>

#### Method 1:

In this method, you are providing Angular with the de-sugared format that needs no further processing. This time Angular would only convert <ng-template> to comments but leaves the content inside it untouched (they are no longer inside any <ng-template> as they were in the previous case). Thus, it will render the content correctly.

To know more about how to use this format with other structural directives refer to this [article](https://www.concretepage.com/angular-2/angular-4-ng-template-example#ngSwitch).

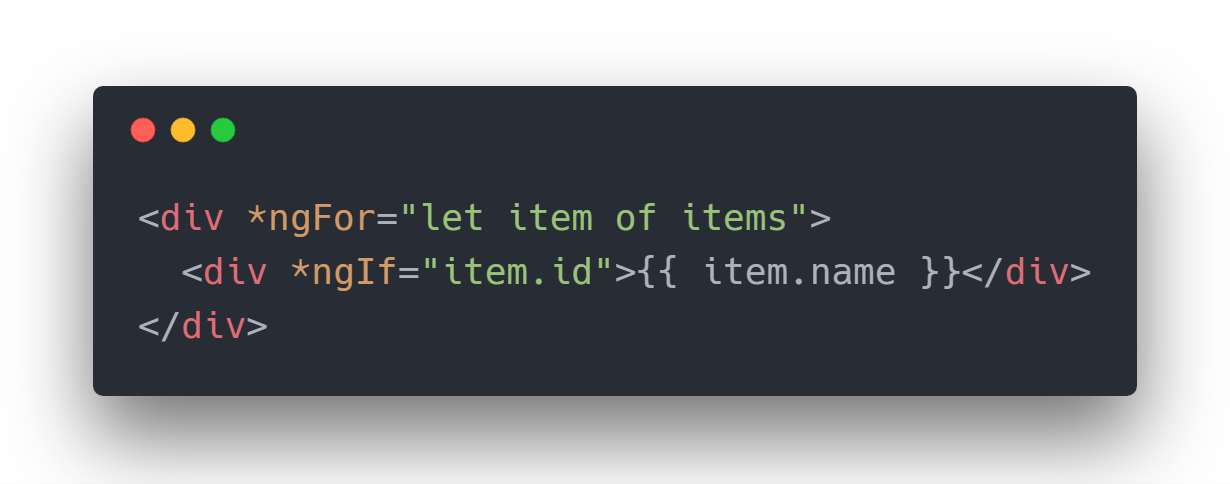
#### Method 2:

This is a quite unseen format and is seldom used (using two sibling <ng-template>). Here we are giving a template reference to the \*ngIf in its then to tell it which template should be used if the condition is true.

Using multiple <ng-template> like this is not advised (you could use <ng-container> instead) as this is not what they are meant for. They are used as a container to templates that can be reused at multiple places. We will cover more on this in a later section of this article.

### 2. <ng-container>

Have you ever written or seen code resembling this:

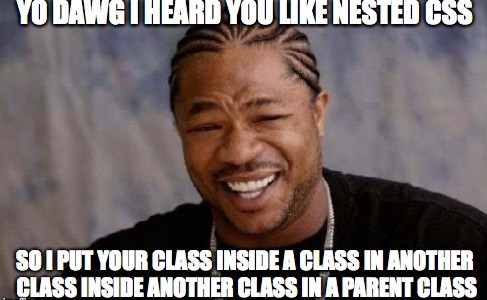
**Example 1**

The reason why many of us write this code is the inability to use multiple structural directives on a single host element in Angular. Now this code works fine but it introduces several extra empty <div> in the DOM if item.id is a falsy value which might not be required.

**Example 1**- Final rendered DOM

One may not be concerned for a simple example like this but for a huge application that has a complex DOM (to display tens of thousands of data) this might become troublesome as the elements might have listeners attached to them which will still be there in the DOM listening to events.

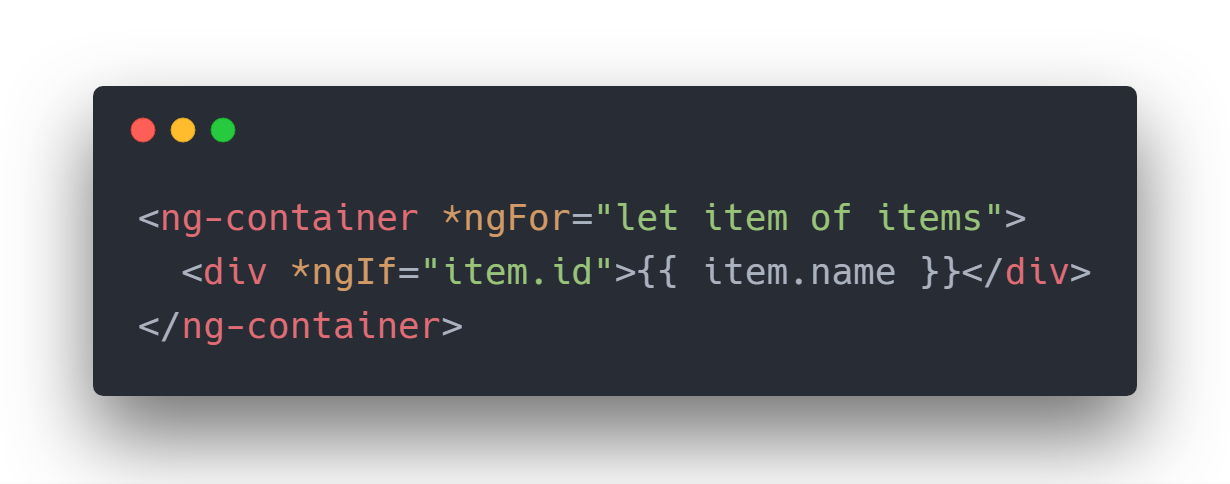
What’s even worse is the level of nesting that you have to do to apply your styling (CSS)!

Image from: [Inside Unbounce](https://inside.unbounce.com/product-dev/making-css-fit/)

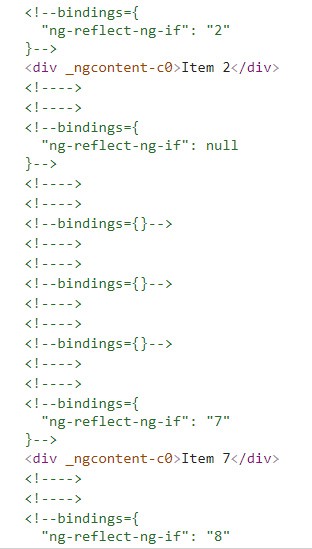
No worries, we have <ng-container> to the rescue!

The Angular <ng-container> is a grouping element that doesn't interfere with styles or layout because Angular doesn't put it in the DOM.

So if we write our **Example 1** with <ng-container>:

**Example 1**with <ng-container>

We get the final DOM as:

Final rendered DOM with <ng-container>

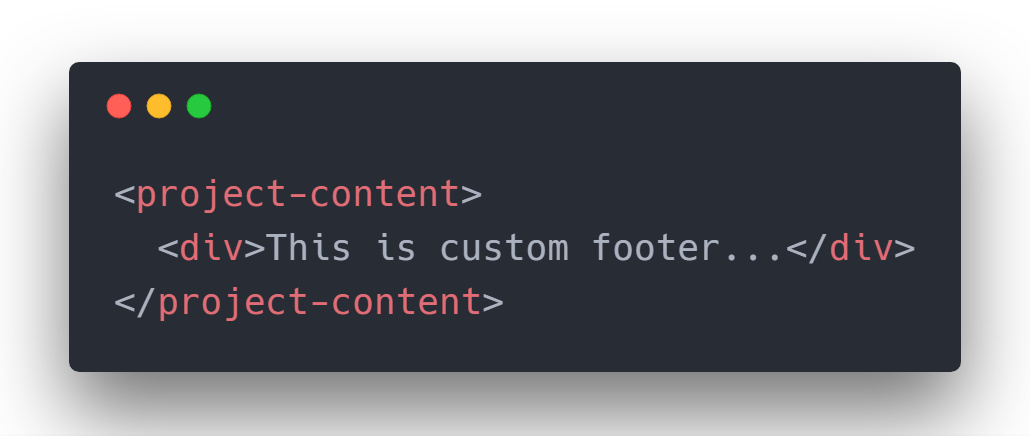
See we got rid of those empty <div>s. We should use <ng-container> when we just want to apply multiple structural directives without introducing any extra element in our DOM.

For more information refer to the [docs](https://angular.io/guide/structural-directives#group-sibling-elements-with-ng-container). There’s another use case where it is used to inject a template dynamically into a page. I’ll cover this use case in the last section of this article.

### 3. <ng-content>

They are used to create configurable components. This means the components can be configured depending on the needs of its user. This is well known as **Content Projection**. Components that are used in published libraries make use of <ng-content> to make themselves configurable.

Consider a simple <project-content> component:

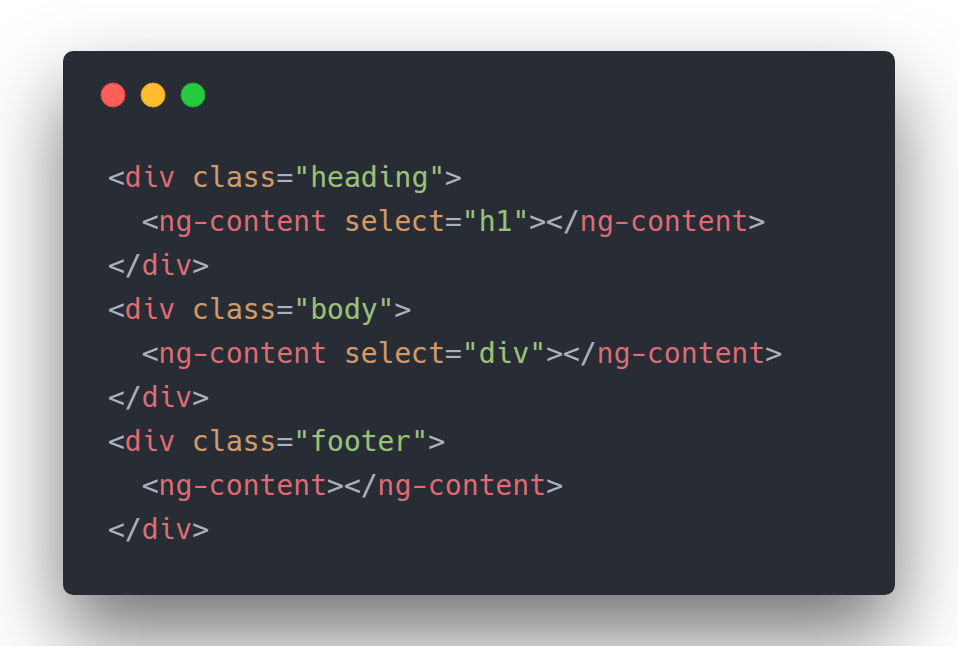
**Example 1**- <project-content> definitionContent Projection with <project-content> component

The HTML content passed within the opening and closing tags of <project-content> component is the content to be projected. This is what we call **Content Projection**. The content will be rendered inside the <ng-content> within the component. This allows the consumer of <project-content> component to pass any custom footer within the component and control exactly how they want it to be rendered.

#### Multiple Projections:

What if you could decide which content should be placed where? Instead of every content projected inside a single <ng-content>, you can also control how the contents will get projected with the select attribute of <ng-content>. It takes an element selector to decide which content to project inside a particular <ng-content>.

Here’s how:

**Example 2**- Multi-content projection with updated <project-content>

We have modified the <project-content> definition to perform Multi-content projection. The select attribute selects the type of content that will be rendered inside a particular <ng-content>. Here we have first select to render header h1 element. If the projected content has no h1 element it won’t render anything. Similarly the second select looks for a div. The rest of the content gets rendered inside the last <ng-content> with no select.

Calling the component will look like:

**Example 2**- Calling of <project-content> component in parent component

### 4. \*ngTemplateOutlet

…They are used as a container to templates that can be reused at multiple places. We will cover more on this in a later section of this article.  
  
…There’s another use case where it is used to inject a template dynamically into a page. I’ll cover this use case in the last section of this article.

This is the section where we will discuss the above two points mentioned before. \*ngTemplateOutlet is used for two scenarios — to insert a common template in various sections of a view irrespective of loops or condition and to make a highly configured component.

#### Template reuse:

Consider a view where you have to insert a template at multiple places. For example, a company logo to be placed within a website. We can achieve it by writing the template for the logo once and reusing it everywhere within the view.

Following is the code snippet:

**Example 1**- Template reuse

As you can see we just wrote the logo template once and used it three times on the same page with single line of code!

\*ngTemplateOutlet also accepts a context object which can be passed to customize the common template output. For more information about the context object refer to the official [docs](https://angular.io/api/common/NgTemplateOutlet).

#### Customizable components:

The second use case for \*ngTemplateOutlet is highly customized components. Consider our previous example of <project-content>component with some modifications:

**Example 2**- Making customizable component, project-content.html

Above is the modified version of <project-content> component which accepts three input properties — headerTemplate, bodyTemplate, footerTemplate. Following is the snippet for project-content.ts:

**Example 2**- Making customizable component, project-content.ts

What we are trying to achieve here is to show header, body and footer as received from the parent component of <project-content>. If any one of them is not provided, our component will show the default template in its place. Thus, creating a highly customized component.

To use our recently modified component:

**Example 2**- Using newly modified component <project-content>

This is how we are going to pass the template refs to our component. If any one of them is not passed then the component will render the default template.

### ng-content vs. \*ngTemplateOutlet

They both help us to achieve highly customized components but which to choose and when?

It can clearly be seen that \*ngTemplateOutlet gives us some more power of showing the default template if none is provided.

This is not the case with ng-content. It renders the content as is. At the maximum you can split the content and render them at different locations of your view with the help of select attribute. You cannot conditionally render the content within ng-content. You have to show the content that is received from the parent with no means to make decisions based on the content.

However, the choice of selecting among the two completely depends on your use case. At least now we have a new weapon \*ngTemplateOutlet in our arsenal which provides more control on the content in addition to the features of ng-content!

 Tweet this to your followers

Email this to a friend

Share this with your friends

Show comments

Countinue reading about

### [Tech](https://www.freecodecamp.org/news/tag/tech/)

* [400+ Online Courses With Real College Credit That You Can Access For Free](https://www.freecodecamp.org/news/370-online-courses-with-real-college-credit-that-you-can-access-for-free-4fec5a28646/)
* [How Being A Mentor Can Be A Double Edged Sword](https://www.freecodecamp.org/news/how-being-a-mentor-can-be-a-double-edged-sword/)
* [My personal favorite Visual Studio Code extensions](https://www.freecodecamp.org/news/visual-studio-code-extensions-ff7f29b71341/)

[See all 2804 posts →](https://www.freecodecamp.org/news/tag/tech/)[](https://www.freecodecamp.org/news/how-to-go-from-idea-to-url-quickly-with-react-js-and-heroku-d94c293c0d9c/)

[#JAVASCRIPT](https://www.freecodecamp.org/news/tag/javascript/)

## [How to go from idea to URL quickly with React.js and Heroku](https://www.freecodecamp.org/news/how-to-go-from-idea-to-url-quickly-with-react-js-and-heroku-d94c293c0d9c/)

9 MONTHS AGO

[](https://www.freecodecamp.org/news/how-to-rock-your-next-time-series-forecasting-project-3930d589f704/)

[#DATA SCIENCE](https://www.freecodecamp.org/news/tag/data-science/)

## [How to rock your next time series forecasting project](https://www.freecodecamp.org/news/how-to-rock-your-next-time-series-forecasting-project-3930d589f704/)

9 MONTHS AGO

freeCodeCamp is a donor-supported tax-exempt 501(c)(3) nonprofit organization (United States Federal Tax Identification Number: 82-0779546)

Our mission: to help people learn to code for free. We accomplish this by creating thousands of videos, articles, and interactive coding lessons - all freely available to the public. We also have thousands of freeCodeCamp study groups around the world.

Donations to freeCodeCamp go toward our education initiatives, and help pay for servers, services, and staff. You can [make a tax-deductible donation here](https://freecodecamp.org/donate).

**Our Nonprofit**

[About](https://freecodecamp.org/about)[Donate](https://freecodecamp.org/donate)[Shop](https://freecodecamp.org/shop)[Sponsors](https://sponsors.freecodecamp.org/)[Email Us](mailto:team@freecodecamp.org)

**Our Community**

[News](https://freecodecamp.org/news)[Alumni Network](https://www.linkedin.com/school/free-code-camp/people/)[Study Groups](https://study-group-directory.freecodecamp.org/)[Forum](https://freecodecamp.org/forum)[Gitter](https://gitter.im/FreeCodeCamp/home)[GitHub](https://github.com/freeCodeCamp/)[Support](https://freecodecamp.org/support)[Academic Honesty](https://freecodecamp.org/academic-honesty)[Code of Conduct](https://freecodecamp.org/code-of-conduct)[Privacy Policy](https://freecodecamp.org/privacy-policy)[Terms of Service](https://freecodecamp.org/terms-of-service)

**Our Learning Resources**

[Learn](https://learn.freecodecamp.org/)[Guide](https://guide.freecodecamp.org/)[Youtube](https://www.youtube.com/freecodecamp)[Podcast](https://podcast.freecodecamp.org/)[Twitter](https://twitter.com/freecodecamp)[Instagram](https://instagram.com/freecodecamp)