I'm not a bot

Reddit and its partners use cookies and similar technologies to provide you with a better experience. By accepting all cookies, you agree to our use of cookies to deliver and maintain our services and site, improve the quality of Reddit, personalize Reddit content and advertising, and measure the effectiveness of advertising. By rejecting non-essential cookies, Reddit may still use certain cookies to ensure the proper functionality of our platform. For more information, please see our Cookie Notice and our Privacy Policy. Hello,I have any issue with flexbox, regarding the flex-grow property.So, I have a navbar and I would like that the first flexbox item would grow twice the size of the other children elements. Now, it doesnt seem to work with flex-grow.nav > header { margin: auto auto; padding: 0px 0px 0px 0px; background-color: red; flex-grow: 2;}Codepen Do you just want the header to take up the full width of the nav container?You have left/right padding on the nav you have also centered the content align-items: center.You might create a container for the paragraph elements (which really should just be links, set to display: block instead) and move the left/right padding from the nav to that container.You can also just overflow the padding on the nav, although that isnt really the best way of doing it. But here is an example anyway.#navbar > header { padding: 35px 0px 35px 0px; background-color: wheat; /* 100% width + container padding */ width: calc(100% + 90px); text-align: center;} Thanks for pointing that out. Why wasnt obvious. Im not seeing the woods because of the tree. Ok, got your solution, tomorrow applying it. Thanks again. slightly off topic, instead of using header tag, it might be more appropriate to use a heading one(h1, h2). Header in html, is more to represent the top element of your page, which can contain various stuff, like logo, title(not to be confused with), and even your navigation bar.Imo, a more correct hierarchy of your tags would be: Heading link A link B link C You could even use a list (for example) to put your navigation links, altho it makes it slightly more complicated to navigate in the CSS rules and will also require additional styling Yeah, i had a similar thought, but going through all of previous challenges and refactor somehow the whole html code.Theyre far not the best work. This topic was automatically closed 182 days after the last reply. New replies are no longer allowed. Baseline Widely available *The flex-grow CSS property sets the flex grow factor, which specifies how much of the flex container's positive free space, if any, should be assigned to the flex item's main size.When the flex-container's main size is larger than the combined main sizes of its flex items, this positive free space can be distributed among the flex items, with each item's growth being their growth factor value as a proportion of the sum total of all the flex items' flex grow factors.Note:It is recommended to use the flex shorthand with a keyword value like auto or initial instead of setting flex-basis on its own. The keyword values expand to reliable combinations of flex-grow, flex-shrink, and flex-basis, which help to achieve the commonly desired flex behaviors. I grow Item Two Item Three.default-example { border: 1px solid #c5c5c5; width: auto; max-height: 300px; display: flex;}.default-example > div { background-color: rgba(0, 0, 255, 0.2); border: 3px solid blue; margin: 10px; flex-grow: 1; flex-shrink: 1; flex-basis: 0;}/* values */flex-grow: 3;flex-grow: 0.6;/* Global values */flex-grow: inherit;flex-grow: initial;flex-grow: revert;flex-grow: revert-layer;flex-grow: unset;The flex-grow property is specified as a single .See . Negative values are invalid. Defaults to 0, which prevents the flex item from growing.This property specifies how much of the remaining space in the flex container should be assigned to the item (the flex grow factor).The main size is either the width or height of the item, depending on the flex-direction value.The remaining space, or positive free space, is the size of the flex container minus the size of all flex items' sizes together. If all sibling items have the same flex grow factor, then all items will receive the same share of remaining space. The common practice is to set flex-grow: 1, but setting the flex grow factor for all the flex items to 88, 100, 1.2, or any other value greater than 0 will produce the same result: the value is a ratio.If the flex-grow values differ, the positive free space is distributed according to the ratio defined by the different flex grow factors. The flex-grow factor values of all the sibling flex items are added together. The flex container's positive free space, if any, is then divided by that total. The main size of each flex item with a flex-grow value greater than 0 will grow by this quotient multiplied by its own growth factor.For example, if four 100px flex items are in a 700px container and the flex items have flex-grow factors of 0, 1, 2, and 3, respectively, the total main size of the four items is 400px, meaning there is 300px of positive free space to be distributed. The sum of the four grow factors (0 + 1 + 2 + 3 = 6) is equal to six. Therefore, each grow factor is equal to 50px ((300px / 6). Each flex item is given 50px of free space multiplied by its flex-grow factor so 0, 50px, 100px, and 150px respectively. The total flex item sizes become 100px, 150px, 200px, and 250px, respectively.flex-grow is generally used alongside the other flex shorthand properties, flex-shrink and flex-basis. Using the flex shorthand property is recommended to ensure all values are set.In this example, the sum of six flex-grow factors is equal to eight, meaning each growth-factor value is 12.5% of the remaining space.HTMLThis is a flex-grow example A, B, C, and F have flex-grow: 1 set. D and E have flex-grow: 2 set. A B C D E FCSS#content { display: flex;}div > div { border: 3px solid rgb(0 0 0 / 20%);}.small { flex-grow: 1;}.double { flex-grow: 2; border: 3px solid rgb(0 0 0 / 20%);}Result When the six flex items are distributed along the container's main axis, if the sum of the main content of those flex items is less than the size of the container's main axis, the extra space is distributed among the size flex items, with A, B, C, and F, each getting 12.5% of the remaining space and D and E each getting 25% of the extra space.SpecificationCSS Flexible Box Layout Module Level 1 # flex-grow-property Hi!In my code the flex-grow doesnt work.I use it at first time, my code more complex than in the flex-box tutorials and examples, thats why the simple examples cant help me..I dont understand which class I have to write flex-grow in.I want that the second div would be twice width than the first.My code: A few things.Firstly, you cant apply a fixed width AND flex-grow.Secondly, flex-grow only works if the parent element has display:flex. In this case the section has display flex but the links do not and the flexgrow divs are children of the linknot the section.Its also not clear what look you are actually going for. Is the right side link supposed to be twice as big as the leftORis the text section of the link supposed to be twice as big as the picture?Finally, flex-grow is NOT the same as widthIf you want precise proportions, use percentages not flex-grow. Thank you for your answer!I corrected some errors as you adviced.I want that the right div (with picture and note too) would be twice big as left.Now, the divs are in a coloumn, i dont know why. They should be in a row.(I want to do it with flex-box, because I want to understand it. Now I am learning :)Te new code: So you want something like this: the parent section has display:flex which means that if the links are supposed to be 1/3 to 2/3 THEY are the ones that need to have the flex-grow valuein this case I just used flex:1 etc. which is the shorthand.BUT, as I said, flexgrow is NOT the same as width. If you want the proportions to always stay the same youll need to use flex-shrink and flex-basis.Note I have moved the text sections to outside the picture divs as it seemed that we waht you wanted. Nice!But I dont understant how could it work and in my site not.There is something mistake.Could you see it, please? in the class box the flex:1; is disabled. Why?I think I followed your example. Maybe the mistake is somewhere else.I copied all here:Maybe easier to see in this way: flex:1 wont work unless the parent has display:flex.Also, you cant relate the size of one element to another unless they are siblings inside the same parent. The parent has the display:flex.In the class galeria and in vateralink, too.class vateralink is the parents,class galeria is parent of vateralink.They both have display:flex. I dont really understand what is the problem with the sizes. Also, you cant relate the size of one element to another unless they are siblings inside the same parent.If box1 and box2 arent in the same element you cant relate their sizes. There is NO CSS method that can do that. Where should I write the size?You cant.as I said.. If box1 and box2 arent in the same element you cant relate their sizes. There is NO CSS method that can do that.Youll have to look into restructuring your HTML. Oh, so my html is wrong! I didnt notice in your example that you put the box elsewhere, I was looking only the CSS.Now I understand! Thank you very much for your help!! One more question:With this flex-box can I declare how many box would be in one row? The forum CSS is closed to new topics and replies. Item 3 3. The `flex-wrap` property is not setThe `flex-wrap` property specifies how flex items are wrapped when they overflow the parent element. If the `flex-wrap` property is not set, the default value is `nowrap`. This means that flex items will not be wrapped, and they will overflow the parent element if they are too large.To fix this issue, you can set the `flex-wrap` property to `wrap`. This will cause flex items to be wrapped when they overflow the parent element.For example, the following code will create a flex container with two flex items. The first flex item is a `div` element with the `width` property set to `50%`. The second flex item is a `div` element with the `width` property set to `50%`.Without the `flex-wrap` property, the two flex items will overflow the parent element. However, if you add the `flex-wrap` property to the parent element and set it to `wrap`, the flex items will be wrapped.The output of this code will be a flex container with two flex items that are wrapped. The first flex item will be on the left side of the container, and the second flex item will be on the right side of the container.4. The `flex-grow` property is not setThe `flex-grow` property specifies how much a flex item should grow when the flex container has more space available. If the `flex-grow` property is not set, the default value is `0`. This means that flex items will not grow at all when the flex container has more space available.To fix this issue, you can set the `flex-grow` property to a positive value. This will cause the flex item to grow when the flex container has more space available.For example, the following code will create a flex container with two flex items. The first flex item is a `div` element with the `width` property set to `50%`. The second flex item is a `div` element with the `width` property set to `50%`. The `flex-grow` property is set to `1` for both flex items.Without the `flex-grow` property, the two flex items would be the same size. However, with the `flex-grow` property set to `1`, the first flex item will grow to fill the entire flex container. The second flex item will remain the same size, but it will be pushed to the right side of the container.There are a few common reasons why `display: flex` might not be working. These include:The `display` property is not set to `flex`.The `flex-direction` property is not set.The `flex-wrap` property is not set.The `flex-grow` property is not set.By checking these properties, you can usually fix the issue and get `display: flex` working as expected.In addition to these common reasons, there are a few other things that can cause `display: flex` to not work. These include:Using the `flex-basis` property with a negative value.Using the `flex-grow` property with a value greater than 1.Using the `flex-shrink` property with a value greater than 1.If you are experiencing problems with `display: flex`, it is a good idea toQ: Why is my flexbox not working?A: There are a few possible reasons why your flexbox might not be working. Here are some of the most common:You forgot to set the `display` property to `flex`. This is the most common mistake people make when using flexbox. To make an element flex, you need to set its `display` property to `flex`.You have multiple flex containers on the same page. Flexbox only works on one level of elements. If you have multiple flex containers on the same page, the inner flex containers will not be affected by the outer flex container.You have conflicting flex properties. Flexbox has a number of properties that can be used to control the layout of flex items. If you have conflicting flex properties, the results will be unpredictable.You are using an outdated browser. Flexbox is a relatively new feature, and not all browsers support it. If you are using an outdated browser, you may not be able to use flexbox.Here are some additional tips for troubleshooting flexbox problems:Use the [Flexbox Playground](to test your flexbox code. This tool can help you identify problems with your flexbox layout.Use the [Flexbox Detective](to see if your browser supports flexbox. This tool can also help you identify problems with your flexbox layout.Read the [W3C documentation on flexbox](for more information on how to use flexbox.Q: How do I make my flexbox items stay in the same order?A: To make your flexbox items stay in the same order, you can use the `order` property. The `order` property determines the order in which flex items are rendered. By default, flex items are rendered in the order in which they appear in the HTML markup. However, you can use the `order` property to change the order of flex items.To use the `order` property, simply add the `order` property to the element that you want to change the order of. The value of the `order` property can be any number from 1 to 65535. The higher the value of the `order` property, the higher the element will be rendered in the flexbox layout.For example, the following code will create a flexbox layout with three flex items. The first flex item will be rendered first, the second flex item will be rendered second, and the third flex item will be rendered third. css.container { display: flex;}.item { flex: 1 1 auto;}To change the order of the flex items, we can add the `order` property to each element. Now, the flex items will be rendered in the following order:1. Item 12. Item 23. Item 3Q: How do I make my flexbox items grow to fill the available space?A: To make your flexbox items grow to fill the available space, you can use the `flex-grow` property. The `flex-grow` property determines how much an element should grow when the flex container has more space than it needs. By default, the `flex-grow` property is set to 0. This means that the element will not grow at all when the flex container has more space.To make an element grow to fill the available space, you can set the `flex-grow` property to 1. This will tell the element to grow until it takes up all of the available space in the flex container.For example, the following code will create a flexbox layout with three flex items. The first flex itemIn this article, we discussed the common reasons why display flex is not working. We covered the following topics:The difference between display flex and other display propertiesThe different flex properties and how to use themCommon mistakes that can prevent flex from working properlyHow to troubleshoot flex issuesWe hope that this article has helped you understand flex and how to use it effectively. If you are still having trouble, please feel free to leave a comment below and we will be happy to help.Here are some key takeaways from this article:Display flex is a powerful layout property that can be used to create a variety of layouts.The flex properties allow you to control the alignment, wrapping, and sizing of flex items.Flex is not a replacement for other layout properties, such as position and float.It is important to understand the different flex properties and how to use them correctly.Flex can be a challenging property to master, but it is well worth the effort. Marcus GreenwoodHatch, established in 2011 by Marcus Greenwood, has evolved significantly over the years. Marcus, a seasoned developer, brought a rich background in developing both B2B and consumer software for a diverse range of organizations, including hedge funds and web agencies. Originally, Hatch was designed to seamlessly merge content management with social networking. We observed that social functionalities were often an afterthought in CMS-driven websites and set out to change that. Hatch was built to be inherently social, ensuring a fully integrated experience for users. Now, Hatch embarks on a new chapter. While our past was rooted in bridging technical gaps and fostering open-source collaboration, our present and future are focused on unraveling mysteries and answering a myriad of questions. We have expanded our horizons to cover an extensive array of topics and inquiries, delving into the unknown and the unexplored. Let the second flex-item grow three times wider than the rest: div:nth-of-type(1) {flex-grow: 1;}div:nth-of-type(2) {flex-grow: 3;}div:nth-of-type(3) {flex-grow: 1;}Try it Yourself The flex-grow property specifies how much the item will grow relative to the rest of the flexible items inside the same container. Note: If the element is not a flexible item, the flex-grow property has no effect.Show demo Browser SupportThe numbers in the table specify the first browser version that fully supports the property. Property flex-grow 29 11 28 9 17 flex-grow: number|initial|inherit; Value Description Play it number A number specifying how much the item will grow relative to the rest of the flexible items. Default value is 0 Demo initial Sets this property to its default value. Read about initial inherit Inherits this property from its parent element. Read about inherit CSS Tutorial: CSS Flexible BoxCSS Reference: flex propertyCSS Reference: flex-basis propertyCSS Reference: flex-direction propertyCSS Reference: flex-flow propertyCSS Reference: flex-shrink propertyCSS Reference: flex-wrap propertyHTML DOM reference: flexGrow property This repository was archived by the owner on Jan 19, 2024. It is now read-only. This repository was archived by the owner on Jan 19, 2024. It is now read-only. You cant perform that action at this time. Flexbox is easily one of the most powerful developments in CSS ever. But, its implementation has been confounding frontend and fullstack devs ever since it was introduced.In the last year alone, 6,000+ new threads were posted on Stack Overflow with questions/issues on CSS flex (Source: Stack Overflow search)!Similarly, Reddit, Quora, and every other platform that developers frequently use are also riddled with devs struggling to get this very powerful but often confusing concept to work for them.We read through a few hundred threads across platforms to find these 7 issues that crop up most frequently when devs think that flex is not working for them.If you think some flex related properties are not working in your code, go through these checks quickly to make sure that you are not making the same errors as those thousands of devs :-)1. Are you using the correct syntax?Make sure you are using the correct and latest CSS syntax throughout the code, particularly if you dont write CSS too often.Its often easy to overlook small errors in CSS selectors or property names because default IDEs dont do a good job at highlighting errors.Youd be surprised how many times we have discovered display:flexbox instead of display:flex while reviewing our own code!Consider using a CSS linting tool like Stylelint, which is great at spotting unintentional errors and enforcing best practices.Fun(?) fact: display: box and display: flexbox used to be the correct syntax in the olden times, way back in 2009 and 2012! Remember that display:flex makes the direct children of the container its applied to, flex items. So, youll have to apply display:flex to the immediate parent of the items you want to distribute using flex.Also, this applies to every level of your div structure. So, if you want to flex an element within a flex element, you must do it twice for both containers. For example, see the code below. Hello World! 1 2 3 3.Are you using flex-grow, flex-shrink and flex-basis correctly?flex-grow: This property defines the ability of a flex item to grow beyond its initial size to fill any available space within a flex container.flex-shrink: This property defines the ability of a flex item to shrink below its initial size to fit within the available space within a flex container.flex-basis: This property defines the initial size of a flex item before any available space is distributed among the items in the container.Here are 2 Stack Overflow examples on how these properties work-Example 1: OutputIn the above code, 50% of 300px, i.e, 150px is immediately assigned to the first item because of flex-basis: 50%;The remaining 150px space is initially empty because the flex-basis of the other two items is 0. Later on, it's divided into all three items equally because each of them has flex-grow: 1, making them 200px, 50px, and 50px respectively.Example 2: In the above code, the flex items have flex-grow: 1 for equal distribution of space but don't have flex-basis and therein lies the point. When you miss to assign a specific flex-basis value to your flex items, its default value becomes auto and the width distribution will be done according to the content size.To fix this issue, also add flex-basis: 0. Flex basis will ensure that there is no default distribution of width and the entirety of it then will be distributed equally among the items because of flex-grow: 1.If you want to understand these properties in much more detail, check out this guide by CSS Tricks. If you are building a UI where certain elements are to be same irrespective of the device and screen, make sure that media queries are not overriding those default properties that need to work on each screen. In the above example, the property flex: 0 0 100%; was overridden by flex: 0 0 calc(50% - 1rem); in the media query and therefore the user was not getting the desired output in the UI.5. Automatic minimum size of Flex itemsAll flex items have an automatic minimum size min-width: auto or min-height: auto on the main axis to avoid shrinking past its content. Remember that it works only for the main axis, so if the flex-direction is row, only min-width will become auto and if the flex-direction is column, then only the min-height become auto.Minimum width or height when set to auto allows flex items to change their size to accommodate the content properly. You can, however, override this default behavior by setting min-width: 0 in row-direction and min-height: 0 in column-direction.Here's an example of this issue- Note: In case you want to convert your email templates to HTML code, check out Kombai For Email.6. If you are using justify-content, remember that default width is auto:One important thing to keep in mind is that if you are not explicitly giving a fixed width to an element, then the default width is auto.An element with the width:auto will take up the smallest needed space for the content and will shrink or expand accordingly to fit its content. In such cases, justify-content will not work as intended because all the available space has already been covered by the content itself and there is no extra space for justify-content to align the flex items. This problem can be solved by giving a fixed width to the element.7. If your issue is happening in some specific browsers/ devicesSometimes, the code you write is correct but it doesn't conform to the ways flex used to work earlier. As a result, it doesn't work as intended on some old browser versions. In the above example, the flex layout is not working on some iPads. This is because it is missing the -webkit- syntax that must be used for iPads that have Safari version 6.1 or less.Here are a few tools and guides that will help you identify and solve cross-browser compatibility issues-Autoprefixer CSS online- Autoprefixer is a tool that automatically adds vendor prefixes to your CSS code. Vendor prefixes are extra code added to CSS properties to ensure they work correctly on different browsers. For example: -webkit-, -ms-, and -moz-.Can I use- This tool shows up-to date browser support tables for various web technologies, including CSS Flexbox.Backwards compatibility of flexbox- A guide by MDN that tells about the history of flexbox and how to write to code that is backwards compatible with the older versions of the browsers.Flexbugs- A GitHub repository that contains lots of flexbox issues, particularly dealing with cross-browser compatibility problems.Wrapping UpThough the article has come to an end, the bugs are still alive. We have covered some of the common flexbox issues developers face and hope this will help you figure out solutions for your codebase.Let us know in the comments if you find this blog post helpful and whether or not it has helped you solve your flexbox related issues.Happy Coding! When you apply a CSS property to an element, theres lots of things going on under the hood. For example, lets say we have some HTML like this: Child Child Child And then we write some CSS .parent { display: flex;} These are technically not the only styles were applying when we write that one line of CSS above. In fact, a whole bunch of properties will be applied to the .child elements here, as if we wrote these styles ourselves: .child { flex: 0 1 auto; /* Default flex value */} Thats weird! Why do these elements have these extra styles applied to them even though we didnt write that code? Well, thats because some properties have defaults that are then intended to be overridden by us. And if we dont happen to know these styles are being applied when were writing CSS, then our layouts can get pretty darn confusing and tough to manage. That flex property above is whats known as a shorthand CSS property. And really what this is doing is setting three separate CSS properties at the same time. So what we wrote above is the same as writing this: .child { flex-grow: 0; flex-shrink: 1; flex-basis: auto;} So, a shorthand property bundles up a bunch of different CSS properties to make it easier to write multiple properties at once, precisely like the background property where we can write something like this: body { background: url(sweettexture.jpg) top center no-repeat fixed padding-box content-box red; } I try to avoid shorthand properties because they can get pretty confusing and I often tend to write the long hand versions just because my brain fails to parse long lines of property values. But its recommended to use the shorthand when it comes to flexbox, which isweird that is, until you understand that the flex property is doing a lot of work and each of its sub-properties interact with the others. Also, the default styles are a good thing because we dont need to know what these flexbox properties are doing 90% of the time. For example, when I use flexbox, I tend to write something like this: .parent { display: flex; justify-content: space-between;} I dont even need to care about the child elements or what styles have been applied to them, and thats great! In this case, were aligning the child items side-by-side and then spacing them equally between each other. Two lines of CSS gives you a lot of power here and thats the neatest thing about flexbox and these inherited styles you dont have to understand all the complexity under the hood if you just want to do the same thing 90% of the time. Its remarkably smart because all of that complexity is hidden out of view. But what if we want to understand how flexbox including the flex-grow, flex-shrink, and flex-basis properties actually work? And what cool things can we do with them? Just go to the CSS-Tricks Almanac. Done! Just kidding. Lets start with a quick overview thats a little bit simplified, and return to the default flex properties that are applied to child elements: .child { flex: 0 1 auto;} These default styles are telling that child element how to stretch and expand. But whenever I see it being used or overridden, I find it helpful to think of these shorthand properties like this: /* This is just how I think about the rule above in my head */.child { flex: [flex-grow] [flex-shrink] [flex-basis];}/* or... */.child { flex: [max] [min] [ideal size];} That first value is flex-grow and its set to 0 because, by default, we dont want our elements to expand at all (most of the time). Instead, we want every element to be dependent on the size of the content within it. Heres an example: .parent { display: flex; } Ive added the contenteditable property to each .child element above so you can click into it and type even more content. See how it responds? Thats the default behavior of a flexbox item: flex-grow is set to 0 because we want the element to grow based on the content inside it. But! If we were to change the default of the flex-grow property from 0 to 1, like this .child { flex: 1 1 auto;} Then all the elements will take an equal portion of the .parent element, butonly if the lengths of their contents are the same. This is exactly the same as writing .child { flex-grow: 1;} and ignoring the other values because those have been set by default anyway. I think this confused me for such a long time when I started working with flexible layouts. I would see code that would add just flex-grow and wonder where the other styles are coming from. It was like an infuriating murder mystery that I just couldnt figure out. Now, if we wanted to make just one of these elements grow more than the others wed just need to do the following: .child-three { flex: 3 1 auto;}/* or we could just write... */.child-three { flex-grow: 3;} Is this weird code to look at even a decade after flexbox landed in browsers? It certainly is for me. I need extra brain power to say, Ah, max, min, ideal size, when Im reading the shorthand, but it does get easier over time. Anyway, in the example above, the first two child elements will take up proportionally the same amount of space but that third element will try to grow up to three times the space as the others. Now this is where things get weird because this is all dependent on the content of the child elements. Even if we set flex-grow to 3, like we did in the example above and then add more content, the layout will do something odd and peculiar like this: That second column is now taking up too much darn space! Well come back to this later, but for now, its just important to remember that the content of a flex item has an impact on how flex-grow, flex-shrink, and flex-basis work together. OK so now for flex-shrink. Remember thats the second value in the shorthand: .child { flex: 0 1 auto; /* flex-shrink = 1 */} flex-shrink tells the browser what the minimum size of an element should be. The default value is 1, which is saying, Take up the same amount of space at all times. However! If we were to set that value to 0 like this: .child { flex: 0 0 auto;} then were telling this element not to shrink at all now. Stay the same size, you blasted element! is essentially what this CSS says, and thats precisely what itll do. Well come back to this property in a bit once we look at the final value in this shorthand. flex-basis is the last value thats added by default in the flex shorthand, and its how we tell an element to stick to an ideal size. By default, its set to auto which means, Use my height or width. So, when we set a parent element to display: flex .parent { display: flex;}.child { flex: 0 1 auto;} Well get this by default in the browser: Notice how all the elements are the width of their content by default? Thats because auto is saying that the ideal size of our element is defined by its content. To make all the elements take up the full space of the parent we can set the child elements to width: 100%, or we can set the flex-basis to 100%, or we can set flex-grow to 1. Does that make sense? Its weird, huh! It does when you think about it. Each of these shorthand values impact the other and thats why it is recommended to write this shorthand in the first place rather than setting these values independently of one another. OK, moving on. When we write something like this .child-three { flex: 0 1 1000px;} What were telling the browser here is to set the flex-basis to 1000px or, please, please, please just try and take up 1000px of space. If thats not possible, then the element will take up that much space proportionally to the other elements. You might notice that on smaller screens this third element is not actually a 1000px! Thats because its really a suggestion. We still have flex-shrink applied which is telling the element to shrink to the same size as the other elements. Also, adding more content to the other children will still have an impact here: Now, if we wanted to prevent this element from shrinking at all we could write something like this: .child-three { flex: 0 0 1000px;} Remember, flex-shrink is the second value here and by setting it to 0 were saying, Dont shrink ever, you jerk. And so it wont. The element will even break out of the parent element because itll never get shorter than 1000px wide: Now all of this changes if we set flex-wrap to the parent element: .parent { display: flex; flex-wrap: wrap;}.child-three { flex: 0 0 1000px;} Well see something like this: This is because, by default, flex items will try to fit into one line but flex-wrap: wrap will ignore that entirely. Now, if those flex items cant fit in the same space, theyll break onto a new line. Anyway, this is just some of the ways in which flex properties bump into each other and why its so gosh darn valuable to understand how these properties work under the hood. Each of these properties can affect the other, and if you dont understand how one property works, then you sort of dont understand how any of it works at all which certainly confused me before I started digging into this! But to summarize: Try to use the flex shorthand Remember max, min and ideal size when doing soRemember that the content of an element can impact how these values work together, too.

Flex grow not growing. Flex grow not working column. Css flex grow vs flex shrink. Tailwind css flex-grow not working. Flex-grow not working.