**Trienuz**

**What is Trienuz?**

Trienuz is a style as well as a standard for organizing a collections of things. When asked to organize a collection of things, people do them in all sorts of ways. So what Trienuz offers is a style and a standard that makes it possible for other people to easily understand how someone else organized a collection of thing. While Trienuz should be confidently used in wherever situations deemed fit, it is know to be excellent in situations where the elements needed to be organized are: stans, dooms, and goons.

**Stans:** A stan is a normal item in the collection. A stan is an item that isn’t a doom or a goon.

**Dooms:** A doom is an item that is a part of another item, although physically separate from the item it is a part of. Take for instance, a JPG file which contains data of a picture displayed in a webpage [HTML file], can be considered a part of the HTML file, even though the HTML and the JPG file are two separate files. And in this case, the HTML file can be considered the stand, while the JPG can be considered the doom.

It is also worthwhile stating that while the example given above is about computer files, Trienuz is not limited to organizing only things related to computers. While the need to organize software source codes, in a better way, is what gave birth to this system, its use isn’t limited to computers only. You should rather feel free to use the system wherever you deem it fit.

However, not only stans can have dooms. Dooms can also have dooms, and goons can also have dooms.

**Goons:** A goon is like a doom. But unlike a doom which would be a part of a single thing, a goon is an item which is part of multiple items. For instance, if multiple webpages [HTML files], make use of a common CSS file, the CSS file should be called a goon, not a doom.

And just like a doom, a goon can also be related to a doom. It can furthermore be related to a goon.

**How it works**

Before delving into a discussion on how Trienuz works, it would be a good idea to quickly explain some terms used in the system.

The paragraphs above discuss what Trienuz is, and under what circumstances it can be used. Now, let’s take a closer look at how the system works

**Trienuz in action**

Imagine you’re asked to develop a simple static website for a school. A website on which the school could make certain information available to its students, and the website should also be a website on which the school could make certain information available to the teachers. What you’d probably do is to create the HTML files for the student pages in the home directory of the project, then create HTML files of the teacher pages in a sub-directory called “Teacher”.

**Project Directory**:

- index.html

- tuition-fee.html

- graduating-students.html

- **Teacher** [sub-directory]

- index.html

- holidays.html

Afterwards, you might go ahead and create the CSS files of each student HTML files in the home directory, then go ahead to create the CSS files of each teacher HTML files in the sub-directory.

**Project Directory**:

- index.html

- index.style.css

- tuition-fee.html

- tuition-fee.style.css

- graduating-students.html

- graduating-students.style.css

- **Teacher** [sub-directory]

- index.html

- index.style.css

- holidays.html

- holidays.style.css

Then you’d probably create the CSS files shared by both the student and teacher HTML files in the home directory of the project.

**Project Directory**:

- index.html

- index.style.css

- tuition-fee.html

- tuition-fee.style.css

- graduating-students.html

- graduating-students.style.css

- global-style-1.css

- global-style-2.css

- **Teacher** [sub-directory]

- index.html

- index.style.css

- holidays.html

- holidays.style.css

While this classic way of organizing items [at least the approach most people take is similar to this, even though it might be slightly different] might be okay for a small collection of things, it may be inadequate when you have to organize a big collection of things. And the reason for this is that the classic method doesn’t help explicitly show all kinds of relationships that may exist between the items that need to be stored. In other words, when dealing with a lot of items, there would be some mental overhead caused by the need to figure out the relationship of items between them. So, what Trienuz offers is a way to explicitly show the relationships between a collection of items being organized. By explicitly helping to show the relationship, it would be more easier to deal with a large collection of items.

So let’s assume we want to organize the files of the project above, using Trienuz, what we’d do is:

Create file “index.html”, “tuition-fee.html”, and “graduating-students.html” in the home directory [In Trienuz we call the main directory *room.*].

[]

Then create file “index.style.css”, and use a string to attach it to the file “index.html”.

[]

Afterwards, you we’d create file “tuition-fee.style.css”, then attach it to file “tuition-fee.html”.

[]

Then we proceed to create file “graduating-students.html”, and attach it to “graduating-students.style.css”.

[]

Up next, we’d create a sub-directory called “Teacher” [We call sub-directories *tooms* in Trienuz.].

[]

Now, in the Teacher toom, we create the file called “index.html”, then create the file called “index.style.css”, and attach file “index.style.css” to “index.html”.

[]

We proceed to do the same for “holidays.html”. We’d create the “holidays.html” file, then create the “holidays.style.css” file. Afterwards, we’d attach the “holidays.style.css” file to the “holidays.html” file.

[]

Lastly, go back to the project’s room [the project’s home directory]. Use a barrier to divide the room into two, but put a door in the barrier so that the right section of the room can be accessed from the main section.

[]

Then in the right section, create the “global-style-1.css” file and the “global-style-2.css” file.

[]

By now, I don’t know if you’ve seen how this system of organization makes more sense. When you walk into the room, you automatically understand how all the items in the room relate to one another. In other words, you wouldn’t even have to invest conscious effort into figuring out how they relate to one another. You automatically know that the items you see in the room are normal items, while the items attached to them are part of them, although they might be psychically separate. Then when you look to the right, you automatically know that the items on the right section are items which are also part of the items in the main section [both standards and attachments], but these items in the right section are related to more than one item in the main section [Let’s assume the items on the right are part of one specific standard/attachment, they’d have been attached to that item. But since they are part of more than one item, we can tie that to a specific item. So rather than attaching them to a specific item, we put them on our right, and we automatically know what that means.]. And when you look towards the back of the room, you automatically know that the inner room is home to a bunch of normal items which shouldn’t be allowed to mix freely with the items in the main room.

By making it possible to explicitly show these relationship, it would much easier to work with a lot of files with ease, since you wouldn’t have to invest conscious effort in figuring out the relationships among items in a place.

By now, you should already have an idea of what Trienuz is. So let’s delve into more details about the system.

**How it works**

**What Inspired The Creation of Trienuz?**

I happpen to be a man of many hats. Software engineering is one of the things I do. Very often, when I come back to a source code, months after I created it, I'd often feel like: what's the file for? How does it fit into the picture? Which files uses this file?, etc. And after facing this kind of problem a good number of times, I felt the need for something like Trienuz, something which can subtly and explicity tell me the relationships between the items in a project dorevtory.

**Terms**

01 Trienuz

This, as you must have known, is the name of the system. Talking about the origin of the name, well, the name was derived from nothing. When deciding what to name this system, I felt I should jiust come up with a random name, and that’s what I did.

02 Trien

This word is a verb. When you trien, it means you organized the items in a storage using Trienuz. And when you trien something, that thing can be said to have been triened.

03 Trienstem

Trienstem is a word that can be used to describe a storage that’s organized using Trienuz.

04 Item

An item is simply a thing that needs to be stored. In computers, it can be a file.

03 Room and Toom

In Trienuz, the term room can be used to refer to the home of a storage. However, it’s not impossible to see it used to refer to a room that is inside of another room. Yes, I understand that I stated that I earlier stated that tooms are rooms that are inside other room, but choosing between room and toom dependends on context. If you just want to generally describe a place in a storage, you should use the world room. But when you not only want to refer to a place in the storage, but also want to indicate that it’s within another room, you can use the word toom.

04 Standard

A standard is an item that needs to be stored in the storage. If an item isn’t an attachement, and it isn’t dependency, and it isn’t a blood, then is has to be a standard.

05 Attachment

An attachment is simply an item that is part of a standard, although physically seprate from it. For instance, a CSS file that contains styling instriction for a partticular HTML file can be considered an attachment of the HTML file.

However, it is also possible for a toom to be an attachment, if the bloods in the toom all happen to be attachments of the standard.

07 Burn

A burn is simply an item that is part of mutiple items [including standards, attachments, bloods, and other burns]. So you can think of a burn as an attachment that is part of mutiple items, rather than a single item.

06 Turn

A burn is simply an area on the right side of a room, where items that are part of multiple standards, attachments, and burn can be placed. Ordinarily, when an item is a part of another item, it should become an attachment of that thing it’s a part of. But since it’s impossible for an item to be stringed to multiple items at once, we just put it on the right side of the room. So once you see something on the right side of the room [in a burn], you automatically know that it’s a part of multiple items.

However, while the examples I gave above shows only one burn, it is possible for a room to have mutiple burns. In fact, there is no limit to how many burns we can have. The number of burns we are going to have, would depend on the items we need to organize.

Take for instance, imagine we have four items. Item C is part of item A and B, and item D is a part of item A, B, and C. How can we trien this collection? Well, to trien this collection, we’d put item A and B in the room, while C stays in a turn. But what about item D? Where are we going to put it? Well, since item D is also a part of item C, it wouldn’t make sense to put it in the same room as item C. What would rather make sense, it to create another turn behind the turn item C is, then put item D in the new turn created.