[SQUEAKING]

[RUSTLING]

[CLICKING]

MICHAEL CUTHBERT: I'm going to switch over to the Jupyter Notebook to talk a review of last class, and then going forward. So what are some of the things we unlocked last class? So Music21 objects that we can use. Shout them out. What's that? Stream. Good.

AUDIENCE: Pitch.

AUDIENCE: Corpus.

MICHAEL CUTHBERT: Pitch, corpus, I heard. Anything else?

AUDIENCE: Duration?

MICHAEL CUTHBERT: Duration. And pitch from duration lets you do what?

AUDIENCE: Note.

MICHAEL CUTHBERT: Note, good. We remember that. And there's one other one I just want to remind you. We're not going to use it very much. It's called converter. Let me make this a little bit bigger. Oh, that's nice for me, too. And sorry. Great. Because converter does the same thing as corpus, but loads any file on your hard drive or anywhere on the web. But since I don't know what's on your hard drive, we're not going to use it that much. Great. And what's something we did last class? What's the problem? So let's all load that Bach piece. Anyone remember what I should type?

AUDIENCE: Corpus.

MICHAEL CUTHBERT: Corpus dot--

AUDIENCE: Parse.

MICHAEL CUTHBERT: Parse. Good. And I said it was Bach BWV 66.6. The only thing you actually need is whatever is necessary to distinguish it from anything else. So you'll sometimes see I'm going to be taking a shortcut there. Good. And how many things were in Bach? Do you remember? Estimate. Six. Oh, good. We have the exact number. Yep. Why? Where are the notes? We have a score object, I'm pretty sure, just to recall. Yep. We have score object. And then what's inside of it, the main thing, four of the six things?

AUDIENCE: Parts.

MICHAEL CUTHBERT: Parts, good. And then inside the part, Hannah, did you-- you had your hand up for the last one. I'll get you in. What's inside of a part?

AUDIENCE: The measure?

MICHAEL CUTHBERT: Measure objects. Good. And what's inside the measure objects?

AUDIENCE: Notes?

MICHAEL CUTHBERT: Notes, and other things, but primarily. So we have this hierarchy, this nested hierarchy. Score, parts, measure, notes. There may sometimes even be another level of hierarchy. For instance, in scores like this where you have two different voices on the same part, there-- what do you call it? There may be a voice object inside of a measure. And if you have a whole collection of scores, we call it something begins with an O-P.

AUDIENCE: Opus.

MICHAEL CUTHBERT: Opus, yeah. So you can have something that's bigger than a score. You can have opuses, opera, technically, an opus that has scores inside it. And then you can put opuses inside opuses, because I didn't want to make anything bigger. Yeah, go ahead, Misha.

AUDIENCE: Can we close the door?

MICHAEL CUTHBERT: Oh yeah, sure, sure. Can you close the door for somebody who can get there? Thanks. Thank you, I appreciate it. Great. So how did we get down to the-- how did we get down to the notes last time? We did something like four thing in score. You don't have to type. This as all review. In Bach if thing, in if-- we remember that isinstance is the Python way of figuring out same thing in score is a stream dot part. We can also just say stream dot stream. If it's some of a stream, then for thing in part in thing in score. I don't want to waste too much time on this. I can do this. If instance thing in part.

Somebody tell me if I type something wrong on this, because I'll probably do it. And there's no tricks on this. For thing in-- what is it? Measure in thing in part, print thing in measure. Hopefully, that works. Yep. So that lets us get down to the lowest level of what's happening here. We have a whole bunch of notes. What are we doing? You'll remember the two representations, the box representation or the salami slice, time wise.

What are we doing? Are we going through all the parts here first, or are we going through one part and then another? Hint. You can always look back at your score. You know the piece and you have the notes. Go ahead. And when you have it, just look up and then shout it out.

AUDIENCE: Part by part.

MICHAEL CUTHBERT: We're going part by part. The soprano C-sharp, B, A, B, C-sharp. Fermata. It's nice to see that fermatas are not in here, so they must be someplace else. E, C-sharp. Yeah, so we're going part by part in this case. Great. And we also saw last class that we could do something. If we knew exactly where something was, we could say Bach 2, which I happen to remember, there's a metadata soprano then alto. So that would be two. I don't know. Let's choose a random measure, and we can make a list out of this. I can't remember if-- I think I demonstrated that. And get what is in here.

And we'll keep seeing that these random other things keep popping up here, like system layouts and so on. So we'll need a better way to get to notes. Questions? This is review, so I went pretty fast. But the important thing is, if I'm reviewing it, it's probably important that we have good understanding before we move on. Yeah, Jake.

AUDIENCE: You had us recurse on the first day? Are we going to use that?

MICHAEL CUTHBERT: So we're going to get to that in just a second. Yep. Good. Yeah.

AUDIENCE: Close that.

MICHAEL CUTHBERT: Sure. Great, great. There we go. Sorry that the blackout blind over there doesn't seem to be working. If you email me after class, I'll remember to put in a facilities request. Or if you email me right now in class, you'll hear a ding. Then I'll put in a facilities request, because that's going to get annoying after daylight savings time starts for this class. Good. Any other things that we want before from the review last time? Yeah. I showed a whole bunch of things on the first class that are not done yet, but we'll be getting to them sooner.

So assume everything on the first class, I think I said at the end, was then immediately relocked. But as far as this recurse thing, we're going to get to it and unlock it in just a few seconds. Great. So any stream can be converted to a list by doing it, so we can get the list of everything that's in the top level score for Bach here. And then we can, yeah, do other things. There are some differences between a stream and a list that are pretty important. So when I say a list, array in other languages, what are some of the things you can do? It's a mutable container. So what some things you can do to it? Yeah.

AUDIENCE: You can add to it.

MICHAEL CUTHBERT: You can add to it. Great. Other things? You usually can add.

AUDIENCE: You can pop things.

MICHAEL CUTHBERT: You can pop things off of it. Great. You can remove things out of it. What are some other things that lists do? Yeah, go ahead.

AUDIENCE: You can change things at certain indices.

MICHAEL CUTHBERT: You can change things at certain indices. Yeah, you can replace one thing with another. Good. Any other things we can do? Yeah, Jason.

AUDIENCE: Find the length?

MICHAEL CUTHBERT: You can find the length. Great. You can find the length, and you can append to it and change the length, all those things. One other thing beginning with IT that you can do. Yeah, John.

AUDIENCE: Iterate on it.

MICHAEL CUTHBERT: You can iterate it. You can put it into a for loop and get things. Great, great. There's one more over here.

AUDIENCE: Splicing?

MICHAEL CUTHBERT: Splicing, yeah. You can get little parts of it, too. So in this way a stream really does work like a list. We can get out, get rid of that metadata. We can get rid of other things here. If I had been smarter 20 years ago, it probably would inherit from list, but it does not. But it has a lot of things that are list-like. But there's some things that you can also do that are a little different. Actually, what was that over here? We went up to Bach 2, 4. Let's get that out here. I think that should be measure three. Let's see. And m equals this, and we'll see what m is. Yep, measure three.

Great. So measure three of the second thing up here, so the third thing, because we're doing 2, 0 index 0, 1, 2, measure three of the alto part. Here is something that I don't think, for all the amazing things MIT CS training does, I don't think it does very, very well. I think that this says measure three of the alto part. And here's the contents of it. Let's just go through and make sure. Remember, pickups are measure what?

AUDIENCE: Zero.

MICHAEL CUTHBERT: Zero. So let's make sure that this is actually measure three. Is that right? Do we get the right measure? The next problem set, the one that we're going to be talking about at the end of class today, it's probably the least time consuming of any of the ones you've had so far, so good. I figure every third one, you need a little bit of a break. So we'll go down on that. I will tell you in advance where people lose points on this problem set, is that they write the code, the code works, and then they go on, and there's a little programming error somewhere that could have easily been fixed by looking at the score during show and checking that your answer was actually what it should be.

So just to have that there, and that's now, my feeling is everybody can get an A plus. We can all get perfects on everything. I don't need to play gotcha games for [LAUGH]. Now there's nothing to mark off. So good. So make sure that you're always doing that, and make sure to do these column sanity checks, that you're getting out from your data what you should be getting in. OK, so these are some things that are similar. One of the things, though, in-- so we have this measure, object n. We'll look. Shape of your brackets does matter.

You have something here. Now, if this was just a normal list, m dot insert 2, we're going to put it position 2. Let's put a note dot note. Let's do one that we can immediately recognize, A double flat 4. We can create an object in here. Actually, we'll be smart. We'll call that n and define it before n equals, so that we can reference it again. Sorry, I'll try not to get this at the very bottom of the screen. Great.

So what should have happened if this were a normal list? Where would you have expected to see that double flat, A double flat? Adam.

AUDIENCE: After the G-sharp.

MICHAEL CUTHBERT: After the G-sharp. Everybody agree? Sometimes everybody agrees when the professor's like, yeah, does it go after two or does it-- I can't remember, but yeah, I'm pretty sure that's right. So it should be somewhere over here.

Why is it here? Let's go back to that alto part, measure 2, and see it's coming after the C-sharp. Does the C-sharp have a fermata on it or not? I just want to make sure we're all on the same place. Yeah, John's nodding his head and John's almost always right on these things, so good, good.

And yeah, so we have it on the last "ya" of that word that I'm not going to say during Lent, right? And good. So it's come in right after there, and yet we said we should put it at 2. What might be happening here? Yeah, John.

AUDIENCE: It's being put at the second beat, 12, 0 index.

MICHAEL CUTHBERT: The second beat, yeah, with the beginning being 0. Now, I told you that's very, very close to the right answer. So great, great. I said at the beginning there was a reason why I didn't like to use beat as a fundamental unit. What was that? Or because of things like 5/8 where the-- fast 5/8, 1, 1 and 2, and a 1 and 2 and a, where beats can change their duration even within a measure. So what's something equivalent? What would durations measured in? I'm going to call on Misha next, but I want to make sure that we have-- yeah, Misha.

AUDIENCE: Yeah, I think it was just said. I think you said you used quarter notes, I assume.

MICHAEL CUTHBERT: Yeah, so quarter length. So in this case, it's exactly the same as beat. So this correct answer in this case, because we're in four quarters. But so we've put it in-- what do we have? Eighth, eighth. So that's all within 0, 0.5, 1, 2. But there might be a problem here. Why is it after the C-sharp? Well, let's do this. We remember some of our iteration for, I'll still, say thing in m, even though they're mostly notes. Print thing and things. New term that we haven't seen, it's offset.

So the offset, just like the duration is measured in quarter lengths, the offset from the beginning of the current container is measured in quarter lengths. So we have here. And we can actually now have two notes at the same time. So you can have multiple things at the same position. That makes it a little bit harder to remove by offset that you want to have, because unlike in an array where there can only be one thing at a particular index, here you can have multiple things at the same index.

This is a simplification. We'll see in a little bit that notes and everything else can have an infinite number of offsets simultaneously. But this is good enough to start here. Yeah, go ahead.

AUDIENCE: So if you do show, it doesn't put them in the same place.

MICHAEL CUTHBERT: If you put it in show, it doesn't. Right, because not, that's a great thing. You're going to find it is easy to create things that are valid musical concepts that cannot be shown by any of our music representation software, our MuseScores, or Finales, or whatever like that. Why? Because if you have a part, do you have two-- This is an ontological question. Do you have two notes that sound at the same time in one part? Or do you have something else that happens when you have two notes at the same time? What other representation might be for two notes, for two things that sound at the same time? I won't say notes. Nobody?

AUDIENCE: Different parts?

MICHAEL CUTHBERT: You would have-- A, you could have it in two different parts, which is what we've normally seen. Or yep.

AUDIENCE: You can have a split, like one person going up and down.

MICHAEL CUTHBERT: Yeah, you can have a split. Yeah, a split. We call those voices in this. So you have two voices at that thing. What's the other thing, another fundamental part of our ontologies?

[PIANO CHORD]

AUDIENCE: Chord?

MICHAEL CUTHBERT: Chord. So in this case, your software wants me to have created a chord object, which is not yet unlocked for us out of these two. So it just does whatever it wants with it. Great question. Super. So keep playing with the limits. We're a little bit behind in time because we're really asking great questions. So I'm going to hold off a little bit on some until we can get a little bit further into some things. So yeah, question. Go ahead.

AUDIENCE: How do you remove?

MICHAEL CUTHBERT: How do you remove? Well, you'll be able to-- you can either remove the thing, and it will find it, Or you can remove something at an offset. And fortunately, all of these other things will be-- all these other options will be revealed to you in the reading before next time we meet together, which is the user's guide chapter on streams. So I think that remove is mentioned in there. It can be a little bit more complicated, as I said, so I want to hold that off for just a little bit.

Let me do another little thing for a second, and we'll jump to some ways that we can make our life easier by not doing for thing and thing, than for next thing and thing, next thing in fixing the previous thing. Obviously, we're going to have some other ideas. So this is one notion, one way a score might be represented in Music21 with two parts. And so within the score in this, its score's length is how much? Just shout it out.

AUDIENCE: Two.

AUDIENCE: Two.

MICHAEL CUTHBERT: Two or?

AUDIENCE: Three.

MICHAEL CUTHBERT: Three, because we have this metadata. And so I've put underneath is the offset of everything. Both parts start at the beginning of the score. So they have offset 0. Within part one, there are two, I'm calling, measures measure 1a, measure 2a. Their offsets are at 0 and 4, which makes sense because somewhere in here, there's a 4/4 time signature. Notice, though, that when we're looking at measure two, measure two's offset is 4, but we start again with offset 0.

So if you want to know how many quarter notes into the piece quarter note 9 happens, you'd have to add 2 plus 4 plus 0. Fortunately, adding 0 is easy, so you get, ah, it's 6 quarter notes into the piece. So this is one, the representation that's happening. Now, when you iterate over the score for all size-- use el, element-- thing, things comes off the tongue better, but el is quicker to type. And when you do this, you'll see that-- we've already seen you get the metadata, you get part one, and then you get part two, and that's all. So you get the three things that are there.

The more powerful way to do this is something called recurse, the recurse method on here. And what this does is it goes through the first thing, and then if there's anything inside the metadata, which there isn't-- it's not a container-- it would go into it. Then part one is a container class. All containers are called streams, but the general term in computational music theory is the container. And then within this container, we have another container, and it will go through these things that are not containers. Sometimes these are called leaves, the leaf node, like in a graph, the thing that has no descendants on it.

And then it will go through the second one. Then it jumps back to part two and goes through each one, and tells you that. So that's going to make life a lot easier. Let's recurse through. Can I move over here? Let's recurse through Bach. For el in Bach dot recurse, print. Let's do el and el's offset. Oops. And now I have to get my scrolling back. Here we go.

So now, we're seeing there's the metadata. There's the soprano part. If we were writing a program, we could keep track of what part we're in at a certain point, and every time we see a new part, do something. And then we see, oh, we're at measures zero. Here we go. Here's all the different things to measure one, measure two. And if I scroll far enough, eventually, we should see-- here we go-- part alto. So that's pretty good.

All this information about what's its offset and things, this can be done, also gotten in a way that just prints. It's not useful for parsing. I would not say parse this. But if you do Bach dot show text, and it just gives you, basically, the same information as that routine, but nicely formatted. So I'll give a second where the current offset is given first, and then the next thing also puts everything like this. Questions on this part?

Now your computer science question. Recurse is what type of graph search? Don't shout it out yet. We'll let everyone think. There are a number of ways of searching through a graph or iterating through a graph. What kind would we call this? I'll give you a hint. The second word is usually first. That wasn't much of a hint. So raise your hand if you're confident that the answer to this now, and go like this if you're not confident that you know the answer. Great.

Could you talk with the person next to you, and especially if you have a couple people who aren't so confident, and share your answer and see if you're in the same boat? Who is now a little bit less-- is still unconfident? Who didn't come to an agreement on your thing? You guys are still in disagreement. No, you're in agreement. Good, good. So let's all shout it out. The kind of search is a--

AUDIENCE: Depth.

MICHAEL CUTHBERT: First search, a depth first search. Good. So we're always going to be following the leftmost-- I should do this in different colors so it's easier to see. The leftmost node that we haven't yet seen, and doing something with that, and then going back up, here, going back up, down. Does that seem to fit other people's computer science things? I have neither a concentration minor, major, or PhD in computer science. So please, if I get something wrong, you all shout to me because you've done more of that than I have. Great. So we have depth first search.

What are some of the things anybody knows from other things that depth first searches are good for? And what are some of the things that they're not good at? So good for anybody who wants to say. Don't worry. It's a safe space.

AUDIENCE: Shortest path problem?

MICHAEL CUTHBERT: Shortest path problem. Good. Shortest path problem. Anybody, do you want to explain? Or anybody else want to explain what that is for those who haven't? Shortest path between two nodes, how to get there. You might not want to do. Something else it might be good for? Yeah.

AUDIENCE: It's just when you're in a scenario where you think you want to look over every possible object.

MICHAEL CUTHBERT: Great. In a scenario where you want to look at every possible object. Now, there's other search methods that also have that property, but this one has that property. So it's already a lot better than for thing and thing than thing and thing, thing and thing. You know that you're going to get to every point. What's something that it might not be good for, or that another search method might be a better way of doing it? Yeah.

AUDIENCE: In the case of breadth first search versus depth first search, if something's close by, there's not a lot of it. Or the tree is super deep. But the answer is super close.

MICHAEL CUTHBERT: Yep.

AUDIENCE: Depth first search will search through every one. Or to go to the depths of the tree.

MICHAEL CUTHBERT: The tree.

AUDIENCE: Just to check in the first couple.

MICHAEL CUTHBERT: Great. Super, super. And we should say what we're comparing. There's lots and lots of searches, but the two key ones, we have depth first search, and it's compared to what?

AUDIENCE: Breadth.

MICHAEL CUTHBERT: Breadth first search. Yep, the breakfast search as I always say. Breadth first search. So that one, what are we going to do if we want to breadth first search in here? What are we going to see first? What's the first node we're going to visit?

AUDIENCE: SC.

MICHAEL CUTHBERT: SC, the score. Good. So, so far, it's exactly the same as the depth first search. Second node we're going to see in this side?

AUDIENCE: Part one.

MICHAEL CUTHBERT: Part one, good. Third node?

AUDIENCE: Part two.

MICHAEL CUTHBERT: Part two. So we're already different. Next node?

[INTERPOSING VOICES]

MICHAEL CUTHBERT: Was one. Yep.

AUDIENCE: For one, isn't BFS better for shortest path searching, rather than DFS?

AUDIENCE: It depends on how you're doing your path. If you're looking for distance one minus path, and you want to find the first one that gets you there, that's better. But you're doing positive length distances in DFS.

MICHAEL CUTHBERT: I'm trying to remember the exact what I remember, which one of these.

AUDIENCE: A star would be even better.

MICHAEL CUTHBERT: Yeah, so we'll get to these particular types of searches later. Controversy in the class, I'll try to remember exactly which one. Do we have a-- but yeah, I'll trust that we'll solve that because we're not going to be doing shortest path yet in this class. But we'll see that there's certain advantages to one versus the other. The other thing, so what's a musical thing that we want to-- how about this? Do all parts start with the same key signature? Breadth first or depth first? Which one is going to be fastest to get you your answer?

[INTERPOSING VOICES]

MICHAEL CUTHBERT: I heard depth and breadth. Who's on depth? Who's on breadth? Yep. OK, good. So we're going to want to have both types of searches available. So recurse will give you always a-- what do you call it-- the depth first search. There's another one that we're going to have. And you're going to see it's the flatten operation. And you're going to see it sometimes in my old slides I haven't figured out how to update, dot flat. But it used to be a property. That is to say, you didn't need the calls called flat. Now it's a method called flatten, I think.

There were two reasons for this. One, if anybody has a new-- uses an IDE, like VS Code or PyCharm, or something like that, which I highly encourage you to do-- it's nothing elite to use VIM anymore in life-- then a lot of those will now go through and call and check every one of the properties at a certain point so it can repeat for you. And flat ended up being for a very, very large score, very space and time sensitive. And so once I upgraded to a thing, I realized, OK, that can't work anymore, that flat. So we had to do this.

I also wanted to make it a few more letters so that people would stop calling flat instead of recurse, just because it was less to type. So now they're much more equal, flatten. And what flatten does is it gives you the option of doing everything first that's at the same offset of everything. So we start with-- I think I have an animation on this-- the metadata, again. Let me get one clef, next clef, both the time signatures, everything that's at the same point. When you have a half note here, we'll get both quarter notes before getting to the other thing.

The other thing that flatten does that's different is you'll see the offsets are now, because it's flattened, they're now all relative to the start of the score. So because of this, they're implemented in different ways. Recurse the breadth first search just has a little pointer that's going around, figuring out where I am, figuring out how to go back up to the next hierarchy level, and walk around, and do this. So it's very, very fast and not very memory intensive.

The flatten, on the other hand, in order to compute where everything should be and which one is actually coming next into this hierarchy, and for historical reasons, flatten will take that same score and put it all into a new score object, a new thing, but with-- it also removes all the parts and stuff. Flatten, there are no parts or anything. There's just the things, just the leaf nodes at the end. So you'll get the clef, the other clef, the time signature, the other time signature, and so on, all in a new stream object.

And this is the first time when I was saying that a note might have more than one offset, because this is the same. Oops. This quarter note here is the same as that quarter note there. So dot offset will be changing as this happens. This is something that will occasionally bite you, but usually not. Which offset does it tell we need to put dot offset? It tells you whatever the offset is in the last stream that you iterated over that has that object in it. So we call that the active site. So it tells you the offset in its active site.

You will find that it's often you're looking at something in a flattened stream that should say flattened up there. And you want to know something about, oh, Professor Cuthbert said, get me all of the notes between offset 5 and 7, and tell me what measure they're in. Shoot. We've gotten all things 5 and 7 from the beginning of the score. The problem is we've lost all the measure information over here. So there is a property on all streams, dot derivation, dot origin, which will take you back to the original, the place where it came from, so the pre flattened stream. And that's something you're going to be working with a lot.

This is an ontological choice I've made on how things are related, how flattened versions of scores have a derivation to the origin of the original score. Now I'll take questions on anything. It's pretty confusing. It's a lot to jump in. Yeah, Adam.

AUDIENCE: Last time, what was BOEHME in the German anthem encoding?

MICHAEL CUTHBERT: What was what?

AUDIENCE: At the very, very top, I said it was the composer.

MICHAEL CUTHBERT: Oh, Bohem. OK, so we're jumping quite a bit back. It's the location of Bohemian, originally, a Bohemian song. That was a pop quiz to see what I remembered. Great. Any other questions on this material specifically, and then on anything? OK, well, let's start by having a little bit of programming exercise time. We have this Bach chorale BWV 66.6. Are there more? Yep, go ahead.

AUDIENCE: So does flatten mutate the object?

MICHAEL CUTHBERT: So flatten mutates-- yes, it mutates something on every object called sites that keeps track of where things are, but it does not change a thing. There were some choices that, now that Music21 is 20 years old almost, there are some things that I would have done differently. And I think it's probably time for somebody who's younger and more ambitious and wants to do something for the next 20 years to come up with things, because I did not listen to our great software engineering Professor Peter Jackson's warnings against mutability in objects as well as I should have. And now, I'm trying to make more things immutable.

But you will find that a lot of things will, by default, not mutate, but then can. And objects can be mutated. So that's a great question. Everyone knows mutable versus immutable objects. You can change, you can not. Lists are--

AUDIENCE: Mutable.

MICHAEL CUTHBERT: What is the equivalent of a list in Python that is immutable?

AUDIENCE: Tuple.

MICHAEL CUTHBERT: Tuple, good. You all say tuple. Here I always learned it as tuple, but tuple. Great. Super. So any other questions before I go on to my question for you about Bach? OK, here is the question for you to answer. And let's do this in groups of two, because it's much more fun to be talking to. Does Bach use more ascending notes or descending notes, descending notes, or about the same?

You might think the piece doesn't start off, Hallelujah, and end all [DEEP MUMBLING], and it doesn't go the opposite, so it must be about the same maybe, but we'll see. So talk. Everyone know who they're going to be paired up with? See pairs. It's easier if you're sitting on think, sit paired. Great. So start talking about how are you going to do this and what which type of--

AUDIENCE: I think I have a quick question. Why does flattening show?

MICHAEL CUTHBERT: Flatten? Why does flatten show?

[INTERPOSING VOICES]

MICHAEL CUTHBERT: Because it's able to-- ah, because recurse is not a stream. Yeah, so I should say, recurse just gives you an iterator around that. You could do recurse thing, and then there's a dot stream that you can put onto it, and that'll give you a stream. But it's just going to give you the same thing you started with, because yeah.

[INTERPOSING VOICES]

MICHAEL CUTHBERT: No, no, no. Then do it then. Yeah, thank you. Thank you. Always say, we're going to talk first about how to do it then implement, but talk.

[INTERPOSING VOICES]

MICHAEL CUTHBERT: It's in class. I don't care however you'd like to do it. But always start talking then implementing. OK I'm hearing the discussion start dying down and hearing some clicks. You can keep talking, but just an informal poll. Which groups are on team flatten? We got one, two, three. Good, good. Which groups are on team recurse, is the best way? Good, good, good. And which groups are in team for thing, and thing, and thing, and thing, and thing? No, just there was a smile with that one.

So keep going. There's ways to keep working on it. I think you'll find there's one that's better than the other. But we'll figure out which one that is.

[INTERPOSING VOICES]

MICHAEL CUTHBERT: And if you're lucky to be sitting next to a group that's on the different team, maybe talk across groups and figure out. We'll dialogue across religions and political views.

[INTERPOSING VOICES]

MICHAEL CUTHBERT: I'm going to interrupt for one second to say something that I've heard. Just a second. Yeah, just one second. Can I interrupt for one second to say something I have heard from two different groups? Matthew. Introduced for two different groups. I have heard this, something like, oh, but think of the memory. We're going to have to keep a reference to the previous note. What is to keep a reference to the previous note? What is the big O memory consideration that we're thinking about?

AUDIENCE: 1?

AUDIENCE: 1?

MICHAEL CUTHBERT: 1. Do we care about? 0 of 1 memory?

AUDIENCE: No.

MICHAEL CUTHBERT: No, we do not. Good. Keep going with that. Just wanted to make sure that we had that we don't care about 0 of 1 operations very often. Occasionally.

AUDIENCE: Is it just this piece?

MICHAEL CUTHBERT: Just this piece. Oh. It was just this. I just heard, is it just this piece or all of Bach? Just this piece is what I'm asking for, but you could do for chorale in corpus dot--

[INTERPOSING VOICES]

MICHAEL CUTHBERT: So you, Derek and Vincent, if you wanted to, there is a for loop you can do above it. And since we've unlocked corpus, that will give you all. Looking good. Looking good. So if anybody's finished and has gotten a thing, oh, a couple questions once you have that. They're not exactly the same. I'll say that. And what are some features in the chorale that explain that? The other thing you could do is do this for all 360 chorales.

[INTERPOSING VOICES]

MICHAEL CUTHBERT: That's a lowercase C in chorales.

[INTERPOSING VOICES]

MICHAEL CUTHBERT: Corpus dot corrals dot Iterator. Just before we go, who is still on team flatten? You used flatten a little bit? OK. And actually, it will say, who used flatten anywhere in the score? And who used recurse anywhere in their score? OK. I'm curious, how did flatten work for getting this?

AUDIENCE: We had to say change or first times flatten, and it's the same for ascending. And then for descending, we're off by 2.

MICHAEL CUTHBERT: Interesting. I'll be curious to see how that happened. Great. So I think we're wrapping up a bit. You can keep working, obviously. What is it? Item 9 on your thing, sound to score, it's one of my favorite things to teach in class because it's really, really fun. But I have made a video so we can save some time outside. And that, I'll add that. That's about a 10 minute to watch. Trying to make it quite a bit easier on this weekend. Try to make it a little bit lighter, so it's just some interesting added things that are happening there.

And for the first time, I'm going to really recommend that you read the Music21 user's guide chapters. There's an optional one, Chapter 26 at the end. You don't need to read it because it's a little bit complex. But if you do read it and get through it, there are some tips in there that can make some aspects of the problem set 3 a little bit easier. So any other questions on what we did?

One last thing. One last thing on this. I always recommend and I really, really recommend a readable code. Write readable code. Write code so that you can do this collaboration and give it to your friends. Write things like that. But one year a while back, the class did decide to go code golfing on this particular thing to see who could write the shortest code to do all this. Oops, sorry.

AUDIENCE: Why did they do that to the eyes?

MICHAEL CUTHBERT: Because back then, we were using Piazza for a discussion board, which used a proportional font. So using eyes made it-- also, it makes it a little bit more inscrutable. So never write code like this. But I do believe that that works if you want it. And your prof won, but see if you can beat that one. Thanks, everybody. [LAUGH]