Project Overview

We are contributing to OpenTimelineIO (OTIO), an open source project owned by Pixar. OTIO allows a team of editors to collaborate effectively. OTIO is an api, file format, and collection of adapters. The OTIO community has a need for more examples and tutorials, so we are creating one here in our github repo (change branch to youtube_demo to see our code).

Want to learn more about the progress that we have made so far? You can read all about it in this doc.

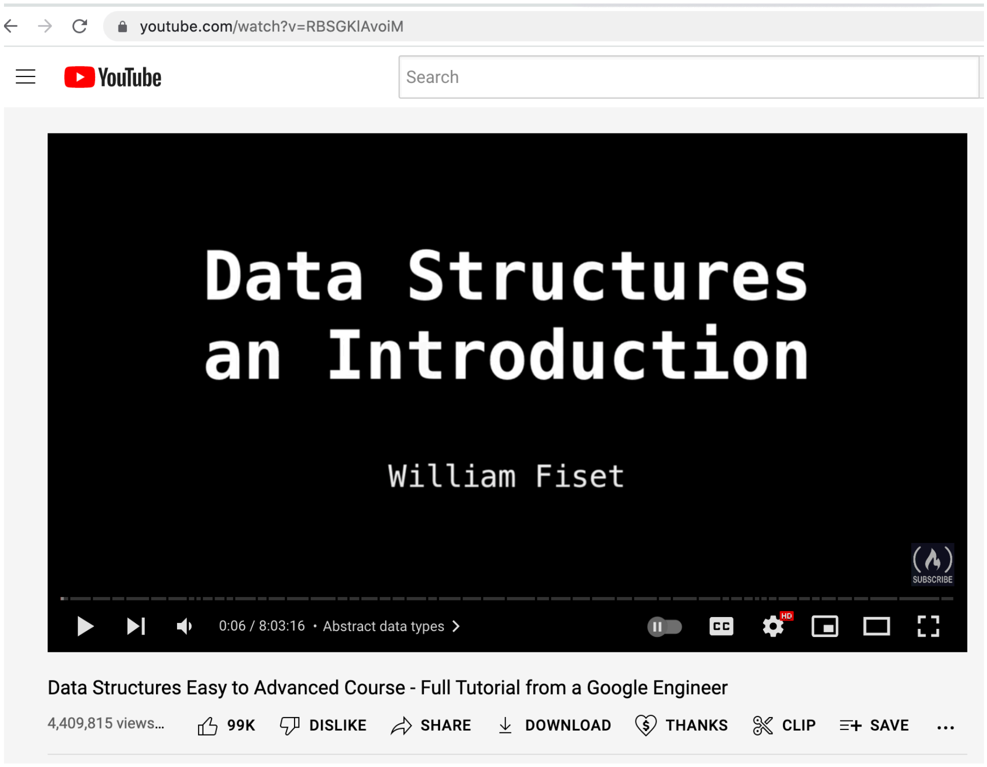
~~Incentive for following along on Twitch~~

~~As an incentive for watching and following along with our open source coding session, we will post a 1 question quiz at the end of every hour. If you can successfully answer 1 quiz per week for 3 weeks in a row, then as a reward you can join Sergio and myself and pair program with us. We can help you get a code commit into our latest open source contribution.~~

Useful Links

- Read this documentation for understanding the key OTIO terms.
- Detailed Installation Instructions for setting up OTIO on a VirtualBox
- Documentation for using youtube-dl in python
- Example of using youtube-dl in python
- OpenTimelineIO calendar
- Regex Cheatsheet
- Regex tool for debugging
- OTIO meeting schedule
- Contributor License Agreement

6/18/22

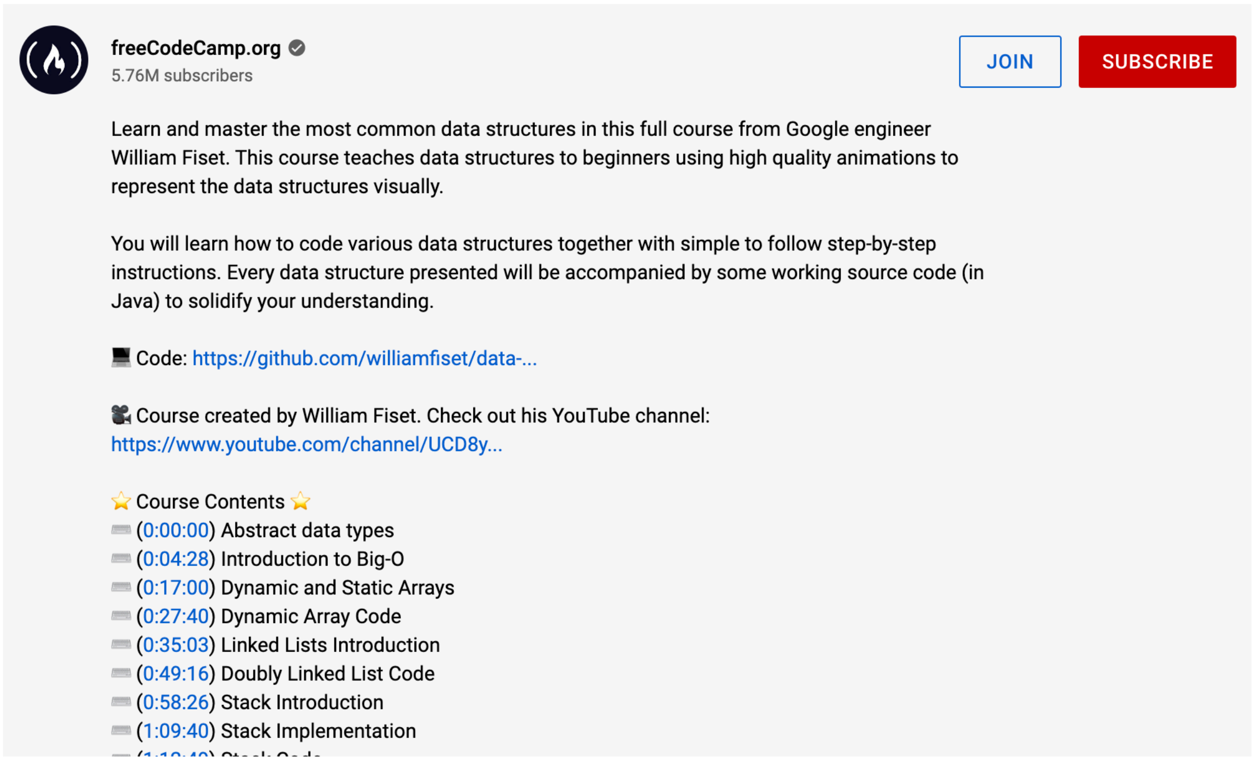

youtube.com/watch?v=RBSGKlAvoiM

Data Structures
an Introduction

William Fiset

Data Structures Easy to Advanced Course - Full Tutorial from a Google Engineer

4,409,815 views...


freeCodeCamp.org
5.76M subscribers

Learn and master the most common data structures in this full course from Google engineer William Fiset. This course teaches data structures to beginners using high quality animations to represent the data structures visually.

You will learn how to code various data structures together with simple to follow step-by-step instructions. Every data structure presented will be accompanied by some working source code (in Java) to solidify your understanding.

💻 Code: https://github.com/williamfiset/data-...

🎥 Course created by William Fiset. Check out his YouTube channel: https://www.youtube.com/channel/UCD8y...

⭐️ Course Contents ⭐️
⌨️ (0:00:00) Abstract data types
⌨️ (0:04:28) Introduction to Big-O
⌨️ (0:17:00) Dynamic and Static Arrays
⌨️ (0:27:40) Dynamic Array Code
⌨️ (0:35:03) Linked Lists Introduction
⌨️ (0:49:16) Doubly Linked List Code
⌨️ (0:58:26) Stack Introduction
⌨️ (1:09:40) Stack Implementation

6/11/22

Continue working on Issue #16 today -- Add support for mkdocs

Goal is to create a draft for Level 1 of our OTIO documentation (explaining OTIO in 5 levels of complexity)

6/4/22

Work on Issue #16 today -- Add support for mkdocs

Try to explain OTIO in 5 levels of complexity: simple to complex.

Level 1

Why? Why does OTIO exist, what problem does it solve? What is it competing with -- older formats, such as EDL (too simple) and fbx(too complex)

Show the difference between .edl file and .otio file. Show the .otio file in OtioView.

Show a youtube video ==> run it through our OTIOExamples script ==> show the final .otio file that is generated.

Input: Video content
Output: .otio file

Talk about OtioView as a companion program/tool that helps us visualize the .otio file.

Explanation how this otio file is then used by editors to make a decision. “As an editor, I would look at XYZ part of the otio file and realize that EFG is off so I would make GHI recommendation to adjust XYZ.”

TODO: Ask the OTIO maintainers for a realistic example that can complete the above scenario.

For the Youtube Demo: “As the Youtube video creator, I can see how all the markers are spaced out, and I’m realizing that we could use another marker at the 4:31 timestamp in the video timeline”

Level 2

Explain this sentence: OTIO is a file format, an API, and a collection of adaptor. (Ideally show a concrete example of each term using our Youtube demo code)

Dive into the structure of the Timeline object (clip, media_reference, track, etc.)
Talk about the API. What does it mean to say that OTIO is an API? Show the youtube demo code, where it makes an API call to compute some useful information about the timeline (total length?). Also show a *write* operation: Inserting markers into the timeline.

Show the documentation for all of the other kinds of API functions that exist.

Collection of adaptors / plugins ==> TODO: more research needed.

Hypothetical example: “We could potentially create an adaptor to read in the Youtube description.”

TODO: Get another concrete example from the OTIO maintainers that we can point to and show some code.

TODO: List out a few common scenarios where adaptor/plugins could prove useful.

Level 3

Outline the technology stack:

- python
- C++
    - make / build commands

Show the simplest possible skeleton of the codebase that makes use of all the key technologies of OTIO. Everything should be "hooked up". Highlight the key files.

- Look at setup.py

Level 4

- Show any relevant design patterns / architectural decisions that they made.

Level 5

- Outline the technology stack.
- Show how the .otio file gets generated from start to finish through the different components of the technology stack ==> put print statements throughout the key steps ==> trail of breadcrumbs

- Show a test and walkthrough the code

OTIO is a ~~**file format**~~, an ~~**API**~~, and a ~~**collection of adaptors/plugins**~~.

3/26/22

Working on Issue #20 today.

01/30/22
Our plan for today: We plan to

01/23/22
Our plan for today: We plan to update a block comment

11:00 we close refactoring if statement.

01/16/21
Our plan for today: We plan to

11:40 General principle about dependency: The more you have to know about x, the worse the dependency is.

12:05 Trying to remove an extra tmp in test code.
12:20 removed the dependency of results, video, description file, etc
01/15/21
Our plan for today: We plan to continue working on package error

ImportError: attempted relative import with no known parent package

2:55 Figuring out how to import a module from the parent directory using relative path.
3:00 We hard coded the path since we wanted to make it work but we plan to refactor it later.

TODO: We noticed a bug: The media_reference is wrong. Gets saved as “tmp/tmp/pvkTC2xIbeY.mp4”

01/09/21
Our plan for today: We plan to continue working on creating a command line argument for skipping download video.

11:40 - We are figuring out how to import the youtube chapters demo module.
12:00 Terms we learn about are module and package.

01/08/21
Our plan for today: We plan to work on creating a command line argument for skipping downloading video and argument parsing.

1:37 - Reading through the argparse python documentation.
2:03 - We figure out that an action is used for the command line flag.
2:30 - We plan to work on refactoring the command line so we can use click.

12/16/21
Our plan for today: We plan to continue working on the issue from 12/05/21

11:21 - We are rereading our notes so we can understand what we did.

11:51 - We commit our changes for organizing top level statements into main function.

12:00 - We add a flag to skip video download and .

12/05/21
Our plan for today: We plan to work on organizing top level statements into the main function.

10:40 Stephan Steinbach gives us an example to run the program without having to shell out.
10:56 We're learning about parsing arguments.

1. https://docs.python.org/3/library/argparse.html
2. https://realpython.com/command-line-interfaces-python-argparse/
3.

11/28/21
Our plan for today: We plan to work on using a library to create a file path.

10:38 Utsab setup his Github personal access token for pushing his changes to Github.
11:13 We had to decide which library to use: os.path.join or PathLib. We decided to use os.path.join because it's simpler for our use case.
11:43 We change file path that was hard code to use

11/27/21
Our plan for today: We plan to start working on the issues.

2:24 Went over a conversation about a writing test that we posted on Slack.
2:40 Working on adding duration for each marker issue.

2;50 Went over how the duration of markers work in OpenTimelineIO.
2:54 Fixing the marker's name.
3:08 We use a regex tool to debug our regex.

11/21/21
Our plan for today: We plan to finalize the issues and ask questions to the OpenTimelineIO community.

11:26 To be more organized we are creating milestones for the issues.
11:50 Post a question about writing unit test on Slack.

Note for next OTIO meeting: Ask them if they can open up a unit test issue for a student to work on.

11/14/21
Our plan for today: We plan to continuing to create issues based on the feedback.

11/07/21
Our plan for today: Continuing looking at the code review and creating issues based on the feedback.

1:59 Utsab has a idea of explaining concepts based on the feedback we received from the code review.
11/06/21
Our plan for today: Review the feedback and update base on the feedback.
1:48 We created an issue for adding a python linter to the codebase.
1:49 Investigate the time module since there can be a timezone issue.

10/31/21
Our plan for today: We did a recap from the community presentation.

What we did well leading up to the presentation:

- Give people props (give credit to anyone who helped you / any prior work you built upon)
- Put some effort into creating a polished presentation
- Listened to their needs consistently, presented what we did, and got more feedback. Rinse and repeat.

10/28/21

- Successfully delivered our presentation.
- Overall, the reception seemed very positive. They said they might want to use this project for their yearly presentation event. They were open to giving us a code review.
- They asked us to post about the project to the slack channel, which Sergio did on Oct 29. We are now waiting on a response.

10/24/21
Our plan for today: We plan to continue working on our presentation slides so we can present to the OpenTimelineIO community.

11:50 We did an architecture diagram for the example project.

10/23/21
Our plan for today: We plan to prepare for our presentation to the OpenTimelineIO community.

10/17/21
Our plan for today: We plan to document what unit tests we should implement.

10:38 Created several Github issues that we plan to write unit tests.

1. https://github.com/utsab/OTIOExamples/issues/12
2. https://github.com/utsab/OTIOExamples/issues/13

10/16/21
Our plan for today: We plan to write unit tests.

1:55 We got Makayla’s local codebase up to date.

2:45 We started writing a unit test for verifying an otio file has the correct number of markers.

10/10/21
Our plan for today: Understand why the markers are scrunched up together.

1:29 One of Utsab's questions is how long is one clip?
1:43 Sirproud commented on Twitch “Ration time(7,4) means frame 7 at fps 4” and to change “seconds to seconds * fps”
1:51 In one of FreeCodecamp videos the timestamp is in a different format.
2:03 Debugging on why it crashes when parsing the number of chapters. There should have been 9 chapters.

2:20 Review some of Regex syntax.

2:40 We test the project with a 2 hour video. In the OpenTimelineIO file it has the wrong chapter title.

10/09/21

Our plan for today: Put the right values into the markers.

2:29 We were able to add markers to the clip. There should be 7 markers in the otio file.

2:44 Updated the ReadMe

2:50 We expected to have 7 markers but we only have 1.

2:56 We solve the issue. The problem was the for loop returns the markers after the first iteration.

2:57 When we view the timeline the markers looks weird. Is there something off?

10/03/21

We are working on this issue today.

Our plan today: Figure out what a “Marker” is and how to use one.

1:21 We are reading OpenTimelineIO documentation so we can learn more about “Marker”.

1:40 We found an example that uses a marker and appends it to a clip. We found it by doing a google search.

1:54 We added a marker to our timeline and now we plan to add multiple markers based on the amount of timestamps.

2:29 We worked on converting the timestamp to seconds.
Saving this code for later:

```
# Append the marker to the clip?

marker = otio.schema.Marker()
marker.marked_range = otio.opentime.TimeRange(
   start_time=otio.opentime.RationalTime(200, dictMeta['fps']),
   duration=otio.opentime.RationalTime()
)

marker.color = otio.schema.MarkerColor.RED

marker.name = "Test Marker!!!"
clip.markers.append(marker)
```

10/02/21

1:18 We are setting up Makayla development environment for OpenTimelineIO example project.

1:53 We ran into an SSL Error when running the OpenTimelineIO example project. Here is the Stackoverflow page. ERROR: Unable to download API page: <urlopen error [SSL: CERTIFICATE_VERIFY_FAILED] certificate verify failed: unable to get local issuer certificate (_ssl.c:1129)> (caused by URLError(SSLCertVerificationError(1, '[SSL: CERTIFICATE_VERIFY_FAILED] certificate verify failed: unable to get local issuer certificate (_ssl.c:1129)')))

2:05 We are updating the readme document so it can reflect what errors we ran into.

2:30 We are giving Makayla a tour to the OpenTimelineIO example project.

09/26/21

2:00 We found a regex to search for three cases.

1. 1:23
2. 12:34
3. 1:23:45

2:39 Later down the line we plan to refactor the code so the code can be more modular and write unit tests for each of the functions.

2:50 We have to refactor the regex because the or operator is making the regex find an empty string.

3:03 We need to figure out how to match the text for the timestamp.
3:10 We successfully figured out how to match both the timestamp and the text of the chapter title using a regex.

09/25/21
1:40 We made a commit for our test that verifies that the timeline was created.
1:52 Later down the line we plan to create an otio adapter to parse the description.
1:59 We plan to use regular expressions to parse the text. The pattern we are looking for is a timestamp (number) with a text.
2:29 Searching how to use python regex.
2:44 We are figuring out how to make regex flexible so we can check multiple digits in a timestamp.

09/19/21

2:40 Our goal for today is to write a test for the media reference.
2:42 A test was created to check if a otio file was created.
2:54 question for the OTIO team: How should we decide which tests to write? Are there high-priority items that should be tested for vs. low priority items that we don't need to test for?

09/05/21

1:30 We are writing tests for processing the youtube video.
1:56 The timeline duration is off by 1 second.
2:01 We figure out why the duration is off by 1. The reason is because the youtube-dl made the duration to 82.
2:15 We had a breakthrough. We were able to display the timeline using otioview.

09/04/21

Goal: Calculate the duration of the Youtube video and successfully import it into the OTIO Timeline object
1:40 We are figuring out how to use a timeline with a single clip by reading the OpenTimelineIO example source code.
2:15 We are able to build and print the timeline data. The timeline didn't have available range but it had source range set to none.

08/22/21
1:30 Utsab's virtual box is not working correctly.
1:44 Utsab was able to get pip to install PySide2 by restarting Virtual Box. The VirtualBox ran out of RAM.
1:57 Utsab is figuring out how to use Github personal access token for authentication.
2:16 Reviewing what we have done so far for the otio example project.
2:30 Rerunning the otio example project.
2:40 We are able to use a json object to get the video description and duration.
2:48 We can't just use the duration value because OTIO requires a RationalTime format.
2:54 We figuring out how to calculate the video total frames. Duration * FPS

08/15/21

2:20 Utsab is setting up the development environment.
2:33 Utsab is setting up the codebase.
2:45 Utsab is setting up Github's new authentication method.(link)
3:00 Utsab is figuring out how to change git

08/14/21

2:05 Utsab is installing Ubuntu in a VirtualBox
2:20 We are figuring out how to download a video using the youtube-dl python package.
2:25 We are searching the youtube-dl source for hints how to download the video
2:27 Lesson: Less is more

08/07/21

2:00 We are figuring out how to install youtube-dl as a python package. We are using python3 for our example.

2:10 We installed the python package for youtube-dl (using pip). Then we imported youtube_dl into our python example.
2:20 Documented how to set up the development environment on our github repo.
2:30 We are able to download an audio file from youtube using the youtube-dl package.

7/17/21

Going through setup instructions:
https://opentimelineio.readthedocs.io/en/latest/tutorials/quickstart.html

1:38: Sergio ran into the following error when running otioview:

```
qt.qpa.plugin: Could not load the Qt platform plugin "xcb" in "" even though it was found.

"Cannot load library
/home/sergio/.local/lib/python3.7/site-packages/PySide2/Qt/plugins/platforms/libqxcb.so:
```

2:00: Switched to Utsab’s computer, and Sergio plans to keep working on the installation errors offline.

2:20: Finished and closed an issue

2:40: Investigated ways to import the Youtube video into OTIO

2:50: Learned that we can import youtube-dl as a python package. We should do this instead of invoking youtube-dl as a command-line process ==> example

How to run the unit tests:
Python3 -m unittest test_youtube_chapters_demo.py

7/11/21

2:20 We ran the example code (youtube_chapters_demo.py) and the test so we can refresh our minds about how it works.

2:35 We had to figure out how to run the youtube chapter demo from within the test. Our approach was to use a system call to run the youtube_chapters_demo.py file.

Quiz: https://forms.gle/nLJK8W9KKtyCjuVw8

How to run the test from command line:

```
python3 -m unittest test_youtube_chapters_demo.py
```

Prior to July 11, 2021, we documented our progress on these slides.

On BREAK until 10:50pm PST