Zucchini Documentation

Release 2.1.1

Zucchini Team

Jan 21, 2023

Contents

Contents:

CHAPTER 1

Zucchini

Zucchini is an automatic grader tool for use in grading programming assignments.

- Free software: Apache Software License 2.0
- Documentation: https://zucchini.readthedocs.io.

1.1 Installation

```
$ pip install --user zucchini
$ zucc --help
```

1.2 Getting Started with Development

After cloning this repo and installing virtualenv, run

```
$ virtualenv -p python3 venv
$ . venv/bin/activate
$ pip install -r requirements.txt
$ pip install -r requirements_dev.txt
$ zucc --help
```

1.3 Features

- Unified grading infrastructure: eliminates maintenance load of ad-hoc per-assignment graders

- Separates test results from computed grades: graders provide test results which are stored on disk, and then zucchini calculates grade based on the weight of each test. That is, graders do not perform grade calculation; they only gather information about students' work
- Simple configuration: update one YAML file and store your graders in git repositories for all your TAs
- Relative weighting: no more twiddling with weights to get them to add up to 100
- Import submissions from Gradescope, Canvas Assignments, or Canvas Quizzes
- No more copy-and-pasting grades and commments: automated upload of Canvas grades and gradelogs
- Flatten (extract) archived submissions
- Gradescope integration: generate a Gradescope autograder tarball for an assignment with one command

1.4 Credits

- Austin Adams (@ausbin) for creating lc3grade, which eventually became zucchini
- Cem Gokmen (@skyman) for suggesting converting lc3grade into a generalized autograder for more than just C and LC-3 homeworks, and creating the initial structure of zucchini
- Patrick Tam (@pjztam) for implementing a bunch of graders, gradelogs, and gradelog upload
- Kexin Zhang (@kexin-zhang) for exploring Canvas bulk submission downloads and for creating the demo downloader, which changed our lives
- Travis Adams (@travis-adams) for nothing

CHAPTER 2

Installation

2.1 Stable release

To install Zucchini, run this command in your terminal:

```
$ pip install zucchini
```

This is the preferred method to install Zucchini, as it will always install the most recent stable release.

If you don't have pip installed, this Python installation guide can guide you through the process.

2.2 From sources

The sources for Zucchini can be downloaded from the Github repo.

You can either clone the public repository:

```
$ git clone git://github.com/zucchini/zucchini
```

Or download the tarball:

```
$ curl  -OL https://github.com/zucchini/zucchini/tarball/master
```

Once you have a copy of the source, you can install it with:

```
$ python setup.py install
```

CHAPTER 3

Usage

Before following this guide, make sure you've installed zucchini as described in *Installation*.

3.1 Grading an Assignment

The following section is written as a zucchini workflow that would be used by a TA in a course that already has a zucchini farm set up (as an example, we will use the sample zucchini farm), that an instructor or TA has already prepared the assignment and linked to it on the farm (as an example, we will use the sample zucchini JUnit assignment), and that student submissions are available in directory (as an example, we will use sample submissions on a git repo).

Note that this tutorial expects that you are on either Linux or OSX, that you have access to the terminal (Terminal.app on OSX), that you have installed a Python distribution that's >=3.4 (we recommend Anaconda for beginners), that you have git installed, that you have JDK 1.8 or higher installed and linked to your path, and that you have gradle installed.

Let's start by installing zucchini

```
pip install zucchini
```

We set up our workspace by entering our identity details:

```
zucc setup
```

Then we add the farm for the metadata repository created by our instructor. We name it `cs1337-fall1970`:

```
zucc farm add https://github.com/zucchini/sample-farm.git cs1337-fall1970
```

Then we make a new directory for our grading and change into it.

```
mkdir zuccsample && cd zuccsample
```

We list the assignments on our farms to find the one we're looking for:

```
zucc list
```

From the output of this, we find that our assignment is called `junit/stacks-queues`. We use `zucc init` to make zucchini pull the assignment configuration into a new directory which will have the assignment's name. We use our farm's name as well as the assignment's name on the farm. Note that detailed information about this assignment, which tests a Stack and Queue implementation using JUnit, can be found on the repository page for the assignment.

```
zucc init cs1337-fall1970/junit/stacks-queues
```

Then, we download the sample submissions:

```
git clone https://github.com/zucchini/sample-assignment-submissions.git
```

Now we change into our assignment directory, and make zucchini load the submissions we just downloaded. Note that in a real workflow, submissions would likely be loaded through LMS integration modules such as Canvas. Also note that the *-d* flag for the path loader is used to make zucchini use the directory name (e.g. *Alice*) as the submitting student's name as well.

```
cd stacks-queues
zucc load path -d ../sample-assignment-submissions/Alice
zucc load path -d ../sample-assignment-submissions/Bob
zucc load path -d ../sample-assignment-submissions/Charlie
zucc load path -d ../sample-assignment-submissions/Dave
zucc load path -d ../sample-assignment-submissions/Eve
```

Then, we start the grading process. This will grade each submission separately and save their results in their folders into the submissions' meta.json files. Once the grading is done, a text editor will open to show the newly updated grades. Hit *:q* close it.

```
zucc grade
```

Now that we're done grading, we want to exports the grades our students received. Note that in a real workflow, this would also likely be done through LMS integration modules such as Canvas, which allow for grades to be saved directly onto students' accounts.

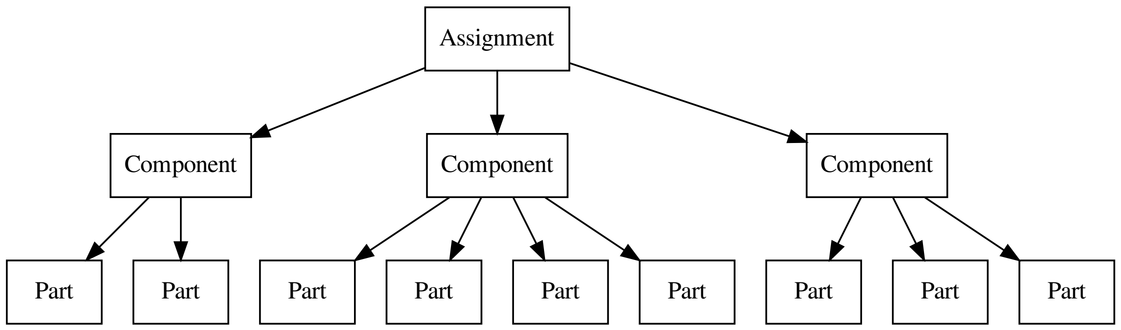
```
zucc export csv > grades.csv
```

And we're done! The grades can be found in the CSV file.

3.2 Creating an Assignment

3.2.1 Anatomy of an Assignment

A Zucchini assignment consists of a list of components, each of which itself consists of a list of parts. Like this:



Zucchini aims to streamline the process of converting a student's submission to a grade in the gradebook, and an assignment instructs Zucchini how to perform this conversion. Indeed, Zucchini downloads submissions, posts grades, and checks due dates for entire assignments, even if they consist of multiple components.

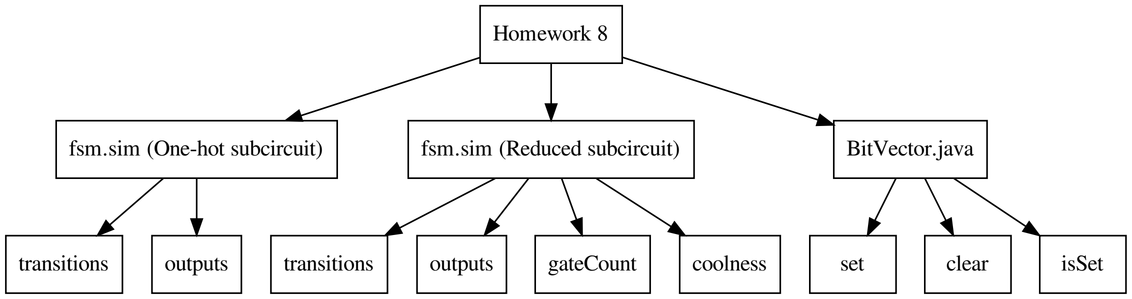
Components represent the smallest pieces of an assignment that Zucchini can grade independently. Usually, this means each independent file in the submission has its own component. Examples of components:

- A test class which tests a particular class in the submission in a JUnit-based grader
- A test suite in a Libcheck-based grader
- A subcircuit in a CircuitSim circuit
- A set of prompts in a prompt grader

Parts represent the smallest result in grading a component that deserves its own weight. We generalized parts because we noticed all of our backends had them. Examples of parts:

- A test method in a JUnit-based grader
- A test in a test suite in a Libcheck-based grader
- A test of a subcircuit in a CircuitSim circuit
- A prompt in a prompt grader

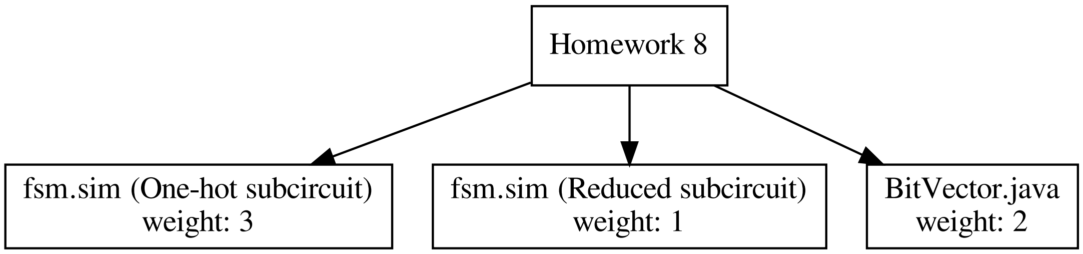
Now, here is a concrete example of the diagram above for a homework with a CircuitSim circuit `fsm.sim` and a Java file `BitVector.java`:



3.2.2 Weights

Zucchini weights components and parts relatively. That is, a component i is worth $\frac{\text{weight}_i}{\sum_k \text{weight}_k}$ of the grade.

So for the following assignment:



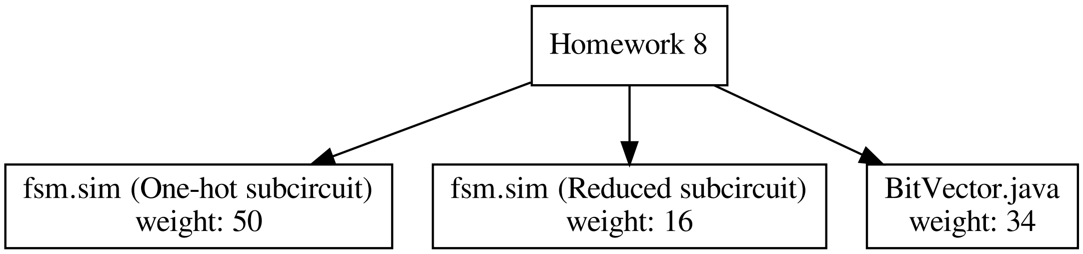
the rubric is actually:

Component	Percent
fsm.sim (One-hot subcircuit)	50%
fsm.sim (Reduced subcircuit)	16.67%
BitVector.java	33.33%

Parts have the same relationship with their parent components. So a part j of a component i is worth $\frac{\text{weight}_i}{\sum_k \text{weight}_k} \times \frac{\text{weight}_j}{\sum_l \text{weight}_l}$ of the grade.

Don't let the decimal points above mislead you: Zucchini calculates grades with rational numbers internally, so you you don't need to worry about floating point screwing up or perfect submissions getting a 99.99 or anything like that (lc3grade had this problem).

We added relative weighting because we didn't enjoy twiddling with weights until they summed to 100. If you do, you can make all the weights add up to 100:



3.2.3 Assignment Configuration

The directory structure for an assignment `my_assignment` looks like:

```
my_assignment/
    zucchini.yml
    grading-files/
        some-grader-jar.jar
        some-grader-file.sh
    submissions/
        Sood, Sanjay/
            meta.json
            gradelog.txt
            files/
                fsm.sim
        Lin, Michael/
            meta.json
            gradelog.txt
            files/
                fsm.sim
```

You need to create only `zucchini.yml` and optionally `grading-files/`. Zucchini will generate `submissions/`. `zucchini.yml` looks like

```yaml
name: Homework X # required
author: Michael Lin # required
due-date: 2018-06-24T18:00:00-04:00
canvas:
  course-id: 2607
  assignment-id: 8685
penalties:
- name: LATE
  backend: LatePenalizer
  backend-options:
    penalties:
    - after: 1h
      penalty: 25pts
components: # required
- name: Finite State Machine # required
  weight: 2 # required
  backend: CircuitSimGrader # required
  backend-options:
    grader-jar: hwX-tester.jar
    test-class: FsmTests
  files: [fsm.sim]
  grading-files: [hwX-tester.jar]
  parts: # required
  - {test: clockConnected,  weight: 1}
  - {test: resetConnected,  weight: 1}
  - {test: enableConnected, weight: 1}
  - {test: outputA,         weight: 5}
  - {test: transition,      weight: 10}
- name: Fully reduced
  weight: 1
  backend: CommandGrader
  backend-options:
    command: "java -cp hwX-tester.jar com.ra4king.circuitsim.gui.CircuitSim fsm.sim"
  files: [fsm.sim]
  grading-files: [hwX-tester.jar]
  parts:
  - text: "banned gates?"
```

(continues on next page)

(continued from previous page)

```
    answer-type: bool
    weight: 2
  - text: "number of incorrect SOP expressions"
    answer-type: int
    answer-range: [0, 5]
    weight: 3
```

You can find a full list of graders at `zucchini.graders`.

3.2.4 Farms

Before Zucchini, grading for us meant hunting down the grader archive on either Slack, Google Drive, or GitHub. Adding to the confusion, sometimes these different sources would get out of sync, forcing TAs to regrade their section all over again. Zucchini offers a solution to this you're probably already comfortable with: git.

TODO: Finish

CHAPTER 4

Config File Format

Configuration files need to be valid YAML files that contain the following fields:

```yaml
name: # Friendly name for the assignment
author: # Author's name (and email if possible)
components:
  - name: # Friendly name for the component
    weight: # Weight of the component (integer)
    files: # Files that need to be copied from the submission folder
    grading-files: # Files that need to be copied from the grading folder
    backend: # Name of the Python class for the grader (e.g. PromptGrader)
    backend-options:
      # The grader backend's options come here - these are listed on the grader's docs
```

CHAPTER 5

Config File Samples

Contents:

5.1 Open File and Logisim Sample

This assignment features two components: a headshot photo which will be opened by the grader and confirmed, as well as a

Sample configuration:

```
name: Headshot and XOR Homework
author: Austin Adams
canvas:
  course-id: 1
  assignment-id: 1
components:
  - name: Headshot image
    weight: 1
    files: headshot.jpg
    backend: OpenFileGrader
    backend-options:
      file-name: headshot.jpg
      prompts:
        - text: Is the image an acceptable image of the student?
          type: boolean
          weight: 1

  - name: XOR circuit
    weight: 3
    files: xor.circ
    grading-files: [hw1checker.jar, brandonsim.jar]
    backend: LogisimGrader
    backend-options:
```

(continues on next page)

(continued from previous page)

```
      logisim-jar: brandonsim.jar
      circuit-file: xor.circ
      prompts:
        - question: Has the student used any banned components?
          type: boolean
          weight: 5
        - question: Has the student successfully connected the inputs to the output?
          type: boolean
          weight: 2
        - question: Does the circuit produce the intended result?
          type: boolean
          weight: 5
```

5.2 Two-Component LC3Test Sample

This assignment features two components, both of which are assembly code files that can be graded using provided lc3test configurations.

Sample configuration:

```
name: LC3 Assembly Homework with Two Components
author: Austin Adams
canvas:
  course-id: 1
  assignment-id: 1
components:
  - name: LC-3 Factorial implementation
    weight: 1
    files: factorial.asm
    grader-files: factorial_test.xml
    backend: LC3TestGrader
    backend-options:
      assembly-file: factorial.asm
      test-file: factorial_test.xml
      runs: 128

  - name: LC-3 Bitvector implementation
    weight: 1
    files: bitvector.asm
    grader-files: bitvector_test.xml
    backend: LC3TestGrader
    backend-options:
      assembly-file: bitvector.asm
      test-file: bitvector_test.xml
      runs: 128
```

5.3 Libcheck Assignment Sample

This assignment has a single .c file being graded using libcheck tests which are individually weighted.

The tests are run on a separate docker container for each student to prevent arbitrary code execution on the grader's computer.

Sample configuration:

```
name: Malloc Homework
author: Austin Adams
canvas:
  course-id: 1
  assignment-id: 1
components:
  - name: malloc()
    weight: 2
    files: my_math.c
    grading-files: tests/*
    backend: DockerWrapperGrader
    backend-options:
      components:
        backend: LibCheckGrader
        backend-options:
          timeout: 5
          build-cmd: make
          run-cmd: ./tests {test} {logfile}
          valgrind-cmd: valgrind --quiet --leak-check=full --error-exitcode=1 --show-
↪leak-kinds=all --errors-for-leak-kinds=all ./tests {test} {logfile}
          tests:
          - name: test_malloc_malloc_initial
            weight: 3
          - name: test_malloc_malloc_initial_sbrked
            weight: 3
          - name: test_malloc_malloc_sbrk_merge
            weight: 3
          - name: test_malloc_malloc_perfect1
            weight: 3
          - name: test_malloc_malloc_perfect2
            weight: 3
          - name: test_malloc_malloc_perfect3
            weight: 3
          - name: test_malloc_malloc_split1
            weight: 3
          - name: test_malloc_malloc_split2
            weight: 3
          - name: test_malloc_malloc_split3
            weight: 3
          - name: test_malloc_malloc_waste1
            weight: 3
          - name: test_malloc_malloc_waste2
            weight: 3
          - name: test_malloc_malloc_waste3
            weight: 3
          - name: test_malloc_malloc_zero
            weight: 3
          - name: test_malloc_malloc_toobig
            weight: 3
          - name: test_malloc_malloc_oom
            weight: 3
          - name: test_malloc_free_null
            weight: 2
          - name: test_malloc_free_bad_meta_canary
            weight: 2
          - name: test_malloc_free_bad_trailing_canary
```

(continues on next page)

(continued from previous page)

```
            weight: 2
          - name: test_malloc_free_empty_freelist
            weight: 2
          - name: test_malloc_free_no_merge1
            weight: 2
          - name: test_malloc_free_no_merge2
            weight: 2
          - name: test_malloc_free_left_merge1
            weight: 2
          - name: test_malloc_free_left_merge2
            weight: 2
          - name: test_malloc_free_left_merge3
            weight: 2
          - name: test_malloc_free_right_merge1
            weight: 2
          - name: test_malloc_free_right_merge2
            weight: 2
          - name: test_malloc_free_right_merge3
            weight: 2
          - name: test_malloc_free_double_merge1
            weight: 2
          - name: test_malloc_free_double_merge2
            weight: 2
          - name: test_malloc_free_double_merge3
            weight: 2
          - name: test_malloc_calloc_initial
            weight: 1
          - name: test_malloc_calloc_zero
            weight: 1
          - name: test_malloc_calloc_clobber_errno
            weight: 1
          - name: test_malloc_calloc_actually_zeroed
            weight: 0
          - name: test_malloc_realloc_initial
            weight: 1
          - name: test_malloc_realloc_zero
            weight: 1
          - name: test_malloc_realloc_copy
            weight: 1
          - name: test_malloc_realloc_copy_smaller
            weight: 1
          - name: test_malloc_realloc_free
            weight: 1
          - name: test_malloc_realloc_toobig
            weight: 1
          - name: test_malloc_realloc_bad_meta_canary
            weight: 1
          - name: test_malloc_realloc_bad_trailing_canary
            weight: 1
```

CHAPTER 6

The Zucchini Architecture

Contents:

6.1 Overview of the Zucchini Architecture

6.1.1 How Farms Work

Right now, zucchini supports grading through two methods - with a locally available configuration file and test suite, or using "farms" - git repos that contain metadata about graders and configurations.

For a given class, for example, the instructor may choose to maintain a single Git repo - a farm - that will contain metadata about all of the course's assignments. Once graders tap into this farm, they will be able to fetch grading configurations and grading files using git automatically and start grading right away without having to download or update any assignment files.

This behavior is managed by the [[Farm Manager|farm-manager]] and the only method of tapping is through Git.

6.1.2 How Loading Works

For grading to be possible, zucchini requires the assignment submissions to be in a precise directory structure. More information about this requirement is available in the [[Directory Structure|directory-structure]] page.

As a result, loaders that implement the [[Loader Interface|loading/loader-interface]]are required to go from arbitrary data sources, like Git and zip / tar archive files, to the zucchini directory structure.

Loaders that are currently available include: * [[Sakai Loader|loading/sakai-loader]] * [[Canvas Loader|loading/canvas-loader]]

6.1.3 How Grading Works

The grading process is managed by the [[Grading Manager|grading/grading-manager]] class, with each rubric item being delegated to a Grader that implements the [[Grader Interface|grading/grader-interface]].

Current implementations of the Grader Interface include:

- [[Prompt Grader|grading/prompt-grader]]
- [[Open-File Grader|grading/open-file-grader]]
- [[LC3Test Grader|grading/lc3test-grader]]
- [[LibCheck Grader|grading/libcheck-grader]]
- [[JUnit Grader|grading/junit-grader]]
- [[Docker Wrapper Grader|grading/docker-wrapper-grader]]

6.1.4 How Exporting Works

Once grading is done, you will need to export your grades. The export process is managed by the [[Export Manager|exporting/export-manager]] class, which gathers the submissions' grades from individual folders into a single dictionary for use by the exporter backends, which have to implement the [[Exporter Interface|exporting/exporter-interface]]. Currently, the supported exporter backends are as follows:

- [[CSV Exporter|exporting/csv-exporter]]
- [[TXT Exporter|exporting/txt-exporter]]

6.2 The Zucchini CLI

Set the user configuration: what's your name? etc. and reset if necessary. Run by default on first run.

```
zucc setup
```

Tap into a Git repo to be able to use configs from it using the tap name that you set:

```
zucc farm add <git-repo-url> <tap-name>
zucc farm remove <tap-name>
zucc farm recache <tap-name> # Equivalent to untapping tap-name and then
tapping its URL again as tap-name
```

List the assignments available for grading

```
zucc list [<tap-name>]
```

Update the taps

```
zucc update [<tap-name>]
```

Load submissions using a loader:

```
zucc load <loader-name> [<loader-parameters>]

# Example with the Sakai loader:
zucc load sakai bulk_download.zip
```

Start grading using a config found on one of the taps (this will automatically update the tap)

```
zucc grade <tap-name>/<assignment-name>
```

Export existing grading results using one of the exporters:

```
zucc export <exporter-name> [<exporter-parameters>]

# Example with the CSV exporter:
zucc export csv hw11.csv
```

6.3 The Farm Manager

There is just a single farm implementation: Git.

6.4 The Loading Layer

Contents:

6.4.1 The Loader Interface

What are our loaders?

6.4.2 The Sakai Loader

How does it work?

6.4.3 The Canvas Loader

How does it work?

6.5 The Grading Layer

Contents:

6.5.1 The LC3Test Grader

How does it work? Config options?

6.6 The Export Layer

Contents:

6.6.1 The CSV Exporter

How does it work?

CHAPTER 7

Contributing

Contributions are welcome, and they are greatly appreciated! Every little bit helps, and credit will always be given.

You can contribute in many ways:

7.1 Types of Contributions

7.1.1 Report Bugs

Report bugs at https://github.com/zucchini/zucchini/issues.

If you are reporting a bug, please include:

- Your operating system name and version.
- Any details about your local setup that might be helpful in troubleshooting.
- Detailed steps to reproduce the bug.

7.1.2 Fix Bugs

Look through the GitHub issues for bugs. Anything tagged with "bug" and "help wanted" is open to whoever wants to implement it.

7.1.3 Implement Features

Look through the GitHub issues for features. Anything tagged with "enhancement" and "help wanted" is open to whoever wants to implement it.

7.1.4 Write Documentation

Zucchini could always use more documentation, whether as part of the official Zucchini docs, in docstrings, or even on the web in blog posts, articles, and such.

7.1.5 Submit Feedback

The best way to send feedback is to file an issue at https://github.com/zucchini/zucchini/issues.

If you are proposing a feature:

- Explain in detail how it would work.
- Keep the scope as narrow as possible, to make it easier to implement.
- Remember that this is a volunteer-driven project, and that contributions are welcome :)

7.2 Get Started!

Ready to contribute? Here's how to set up *zucchini* for local development.

1. Fork the *zucchini* repo on GitHub.
2. Clone your fork locally:

   ```
   $ git clone git@github.com:your_name_here/zucchini.git
   ```

3. Install your local copy into a virtualenv. Assuming you have virtualenvwrapper installed, this is how you set up your fork for local development:

   ```
   $ mkvirtualenv zucchini
   $ cd zucchini/
   $ python setup.py develop
   ```

4. Create a branch for local development:

   ```
   $ git checkout -b name-of-your-bugfix-or-feature
   ```

   Now you can make your changes locally.

5. When you're done making changes, check that your changes pass flake8 and the tests, including testing other Python versions with tox:

   ```
   $ flake8 zucchini tests
   $ python setup.py test or py.test
   $ tox
   ```

   To get flake8 and tox, just pip install them into your virtualenv.

6. Commit your changes and push your branch to GitHub:

   ```
   $ git add .
   $ git commit -m "Your detailed description of your changes."
   $ git push origin name-of-your-bugfix-or-feature
   ```

7. Submit a pull request through the GitHub website.

7.3 Pull Request Guidelines

Before you submit a pull request, check that it meets these guidelines:

1. The pull request should include tests.
2. If the pull request adds functionality, the docs should be updated. Put your new functionality into a function with a docstring, and add the feature to the list in README.rst.
3. The pull request should work for Python 3.4-3.8, and for PyPy. Check https://travis-ci.org/zucchini/zucchini/pull_requests and make sure that the tests pass for all supported Python versions.

7.4 Tips

To run a subset of tests:

```
$ python -m unittest tests.test_zucchini
```

CHAPTER 8

Credits

8.1 Development Lead

- Zucchini Team <team@zucc.io>

8.2 Contributors

None yet. Why not be the first?

CHAPTER 9

History

9.1 0.1.0 (2017-12-17)

- First release on PyPI.

CHAPTER 10

Indices and tables

- genindex
- modindex
- search