
Cannoli Programming Language



Cannoli is a compiler for a subset of Python 3.6.5 and is designed to evaluate the language features of Python that negatively impact performance. Cannoli is written in Rust and also compiles Python to Rust. The use of Rust as the intermediate representation was chosen for performance purposes and to avoid writing a garbage collector. Cannoli was developed as work for a Master's Thesis at Cal Poly - San Luis Obispo.

Python Support

Cannoli supports a subset of Python 3.6.5, its current state omits many features that could not be completed during the duration of the thesis. The main omissions are exceptions and inheritance. Standard library support is also incomplete but covers numerous proofs-of-concepts that could be applied to other types and modules (see Cannolib).

Optimizations

Cannoli supports two major optimizations that come as a result of applying restrictions to the language. Restrictions are placed on the Python features that provide the ability to delete or inject scope elements and the ability to mutate the structure of objects and classes at run time. The corresponding feature branches are `scope-opts` and `class-opts`. The optimizations are built on top of each other, therefore the `class-opts` branch is a superset of the `scope-opts` branch. In general, the `class-opts` branch yields a performance increase of over 50% from the `master` branch.

Thesis Paper

More information on the results and implementation details of Cannoli can be found in the thesis paper.

:point_right: *Leave the Features: Take the Cannoli* - Jonathan Catanio

How to Run

- Install Rust by following the instructions on their official installation guide. Cannoli is both compiled with Rust 1.24.0 and outputs Rust 1.24.0 code. Changing versions with the Rust toolchain can be done with the `rustup` utility.

-
- Build the project by running `cargo build` or `cargo build --release` in the project's root directory. This will create a `target` directory with `debug` or `release` subdirectories containing the executable.
 - Compile a Python file by executing the command `./target/release/cannoli [src.py]`

Executing the Compiled Python

Compiling with Cannoli outputs a `main.rs` file that can be used in a standalone Rust crate. Ideally the Cannoli compiler would utilize `rustc` to output a binary but this wasn't done. The steps on how to run this compiled file are as follows:

- Create a new crate with the command `cargo new --bin sandbox`. Where `sandbox` is the name of the crate.
- Move the `main.rs` file into the new crate's `src` directory.
- Update the `Cargo.toml` file to include `cannolib` as a dependency, this is Cannoli's standard library. This can be done by linking to a git repository or to a relative crate.

```
1 [dependencies]
2 cannolib = { git = 'https://github.com/joncatanio/cannolib', branch = '
  master' }
```

Would include the master branch of the Cannolib git repo in the crate. - To add debugging and all optimizations to release mode, also include these lines in the `Cargo.toml`:

```
1 [profile.release]
2 debug = true
3 codegen-units = 1
```

- Finally, build the project from the crate's root directory with `cargo build --release` and run the executable with the command `./target/release/sandbox`.

Cannolib

Cannolib provides library support for Cannoli, including a number of types and modules that offload work that would have otherwise been done by the Cannoli compiler. Cannolib provides the implementation of the overall type system as well as built-in functions similar to those defined in the Python library.