

User Manual

Contents

1	Creating a New Project or Accessing an Existing One	3
2	Uploading a Code File	3
3	Adding or Removing Code Cells	3
4	Running your code	4
5	Uploading a Test File	4
6	Testing Your Code	5
6.1	Manual Testing	5
6.2	Automatic Testing	5
7	Setting Configurations	6
7.1	Manual Configuration	6
7.2	Configuration File Import	6
7.3	Configuring the arguments and return types	7
8	Saving and Uploading a Local Project	7
9	Choosing the Program Output	8

10 Proper closing of containers and Podman system	8
11 System Limitations	9
11.1 Code Limitations	9
11.2 Operating System Limitations	9

1 Creating a New Project or Accessing an Existing One

Upon launching the application via the desktop icon, the user is presented with the Home Page. Two options are available:

- **Create a New Project**
- **Open Existing Project**

At any time, the user can return to the Home Page by clicking the logo icon located in the sidebar.



Figure 1: Logo Button

2 Uploading a Code File

After proceeding past the Home Page, the user is directed to the **Code Page**, which includes two initial code cells and a sidebar.

Each code cell features a dropdown menu in its header, which contains an **Upload Code** button. Clicking this button allows the user to upload local code files with supported extensions. Upon upload, the system automatically detects the programming language and updates the syntax highlighting and configuration accordingly.

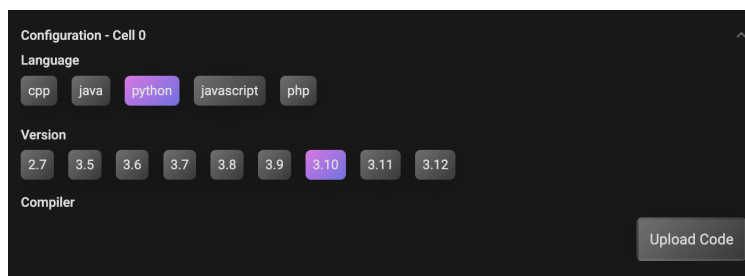


Figure 2: Code Cell Configuration Example

3 Adding or Removing Code Cells

If additional code cells are needed beyond the initial two, the user can click the **Add File** button in the sidebar. Each click adds a new code cell to the right side of the interface.

To remove a code cell, open its header dropdown and click the **Delete** button.

Note: Deletion is only possible when more than two code cells are present on the screen.

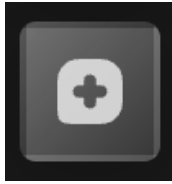


Figure 3: Add Cell Button

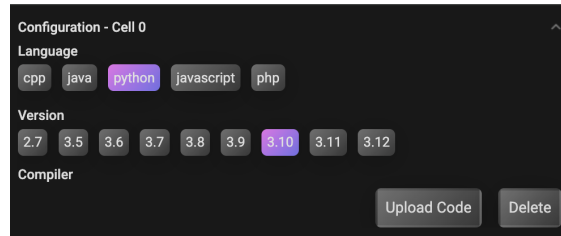


Figure 4: Cell Configuration with Delete Button

4 Running your code

After setting all system parameters, the user can run their code by simply pressing the **Run** button. All cells run in parallel and results will be displayed after all cells are done running.

The user has the option of setting a timeout for running their code from the Settings page. If not specified, default is one minute.

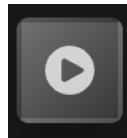


Figure 5: Run Button

5 Uploading a Test File

For manual testing, users can upload a test file containing input-output test cases. The file must be in `.txt` format and should follow the required structure as shown below:

Input

1,2
3,4
5,6

Output

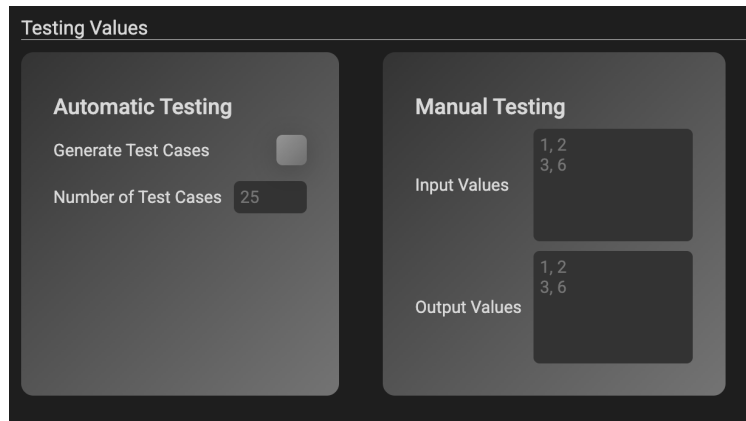
6
7, 9
8

- The **Input** section consists of pairs of values separated by commas, each line representing a separate test case.

- The **Output** section provides the expected results corresponding to the input values.

6 Testing Your Code

In order to set the desired type of testing for their code, the user can access the Testing section in the Settings Page.



The screenshot shows a 'Testing Values' section with two main panels. The 'Automatic Testing' panel on the left includes a 'Generate Test Cases' toggle switch (currently off) and a 'Number of Test Cases' input field with the value '25'. The 'Manual Testing' panel on the right contains two text input areas. The 'Input Values' area has two lines of text: '1, 2' and '3, 6'. The 'Output Values' area also has two lines of text: '1, 2' and '3, 6'.

Figure 6: Testing Section in Settings Page

6.1 Manual Testing

Users can manually input test cases by specifying input and output values. Each row corresponds to a single test case. For example, to define three test cases, enter three rows—each with its own input/output pair.

6.2 Automatic Testing

If the user prefers autogenerated test cases, they can enable the **Generate Test Cases** option and specify the desired number. If not specified, the default is 25.

Note: Automatic testing and manual testing cannot be performed simultaneously.

7 Setting Configurations

7.1 Manual Configuration

Configuration options such as language, version, and compiler can be set manually via the header of each code cell. Before executing the code, each cell must have a signature defined via the **Signature** tab in the Settings Page.

All code cells must have compatible argument and return types. Additionally, the user can either:

- Write their own `main()` function and click the run-as-is button, or
- Specify the name of the method to execute

If a cell uses additional libraries (e.g., `numpy`, `pandas` for Python), these must be specified in the **Specs** input field for that cell.

7.2 Configuration File Import

Users can also upload a configuration file via the **Import** button in the sidebar. The file must follow the application's required format, as shown in the example below:

```
"codeCells": [  
  { "codeText": "code sample 1", "language": "php", "version": "php:5.6", "compiler":  
    { "codeText": "code sample 2", "language": "cpp", "version": "", "compiler": "clang"  
  }  
]
```

- Each code cell is represented by an object within the `codeCells` array.
- The `codeText` field contains the code sample.
- The `language` specifies the programming language (e.g., PHP or C++).
- The `version` indicates the version of the language (e.g., PHP:5.6), if needed.
- The `compiler` specifies the compiler to be used (e.g., `clang` for C++), if any.

7.3 Configuring the arguments and return types

For the system to work properly, the user needs to specify the arguments the function takes and the return type of the function. Even if your code is loosely typed, the return type needs to be clearly specified. For the arguments, the system accepts the following primitive types: int, float, double, string, char, bool. Additionally, you can use lists or nested lists of this types. The format for lists is this: list[type]. You can nest however many [] as you need. Examples: list[int], list[[bool]], list[[[string]]]

8 Saving and Uploading a Local Project

To save a project locally, click the **Save** button in the sidebar. The saved project will follow a specific structure, as shown in the example below:

```
{
  // information for code cell configuration
  "codeCells": [
    { "codeText": "code sample 1", "language": "php", "version": "php:5.6", "compiler": "php",
      { "codeText": "code sample 2", "language": "cpp", "version": "", "compiler": "clang"
    },
    // information for cell signature and testing
    "finalValues": {
      // cell signatures
      // return and args only need to be specified for the first one
      "cells": [
        {
          "name": "sumNumber",
          "args": ["int", "int", "int", "int"],
          "return": "int",
          "specs": ["numpy", "pandas"],
          "run_as_is": false
        },
        {
          "name": "",
          "args": "",
          "return": "",
          "specs": ["pandas"],
          "run_as_is": true
        }
      ],
      // manual testing values
      "input": [],
```

```

    "output": [],
    "timeout": "",
    // automatic testing
    "generate_test_cases": true,
    "test_cases_signature": "",
    "test_cases_count": ""
  }
}

```

To reopen or upload a local project:

- Click **Open Existing Project** from the Home Page, or
- Use the **Import** button in the sidebar.

Alternatively, users may manually create a project folder according to the required format and import it into the application.

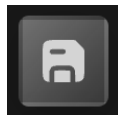


Figure 7: Save Button



Figure 8: Import Button

9 Choosing the Program Output

Output display options can be customized via the **Overlay** tab in the Settings Page. Users can choose which aspects of the program output they want to view.

10 Proper closing of containers and Podman system

As old containers are removed when the application starts, when the user finishes using the application the containers are left hanging. In order to properly remove the containers, the application should be opened again.

Another system that is will be continuously running is the Podman Virtual Machine. Once the application is started once, the Podman VM will be running until it is either stopped or the computer is closed. In order to stop Podman and avoid unintended resource utilization, the user has to run the command "podman machine stop" in a terminal.

11 System Limitations

11.1 Code Limitations

- For the Java Language, lists of strings are not fully supported. The system will work correctly if the strings do not contain the `'` character. Also, the function needs to be static and it shouldn't be written in a class; just the function is enough. Additionally, the function can't be named `main`.
- For the CPP Language, the only array-like structure supported is `vector`. Also, the namespace is not specified, so it needs to be `std::vector` or `std::string`

11.2 Operating System Limitations

- **MacOS/Linux:** Additional configuration may be necessary.
- **MacOS Specific:** Depending on the OS version and chip architecture, users may need to downgrade to an older version of Podman (recommended: version below 5.0).