

# Marimo

# Marimo

🌐 24 languages ▾

Article [Talk](#)

[Read](#) [Edit](#) [View history](#) [Tools](#) ▾

From Wikipedia, the free encyclopedia

**Marimo**<sup>[a]</sup> (also known as **Cladophora ball**, **moss ball**, **moss ball pet**, or **lake ball**) is a rare growth form of ***Aegagropila linnaei*** (a species of filamentous [green algae](#)) in which the algae grow into large green balls with a velvety appearance.

The species can be found in a number of lakes and rivers in Japan and [Northern Europe](#).<sup>[1]</sup> Colonies of marimo balls are known to form in [Japan](#) and [Iceland](#), but their population has been declining.<sup>[2]</sup>

## Classification and name [\[ edit \]](#)

Marimo were first described in the 1820s by [Anton E. Sauter](#), found in [Lake Zell](#), [Austria](#). The genus *Aegagropila* was established by [Friedrich T. Kützing](#) (1843) with *A. linnaei* as

### Marimo




Marimo in [Lake Akan](#) in Japan



**Scientific classification** 

# Marimo

Reactive, Git-friendly, Notebooks

 **batteries-included:** replaces jupyter, streamlit, jupyter, ipywidgets, papermill, and more

<https://marimo.io/>

# What does it try to solve?

- <https://leomurta.github.io/papers/pimentel2019a.pdf>

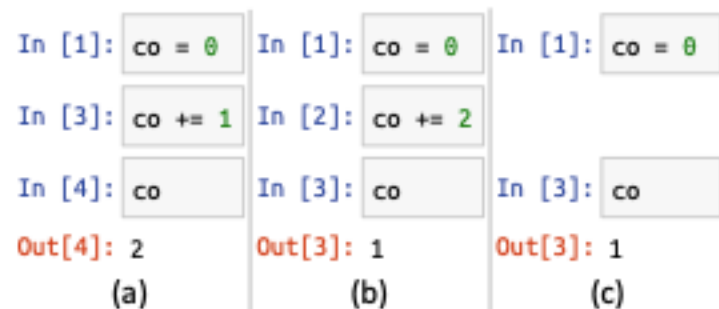


Fig. 2. Three types of Hidden States: (a) Re-execution; (b) edited cell; (c) removed cell.



Fig. 13. Distribution of cell reproducibility.

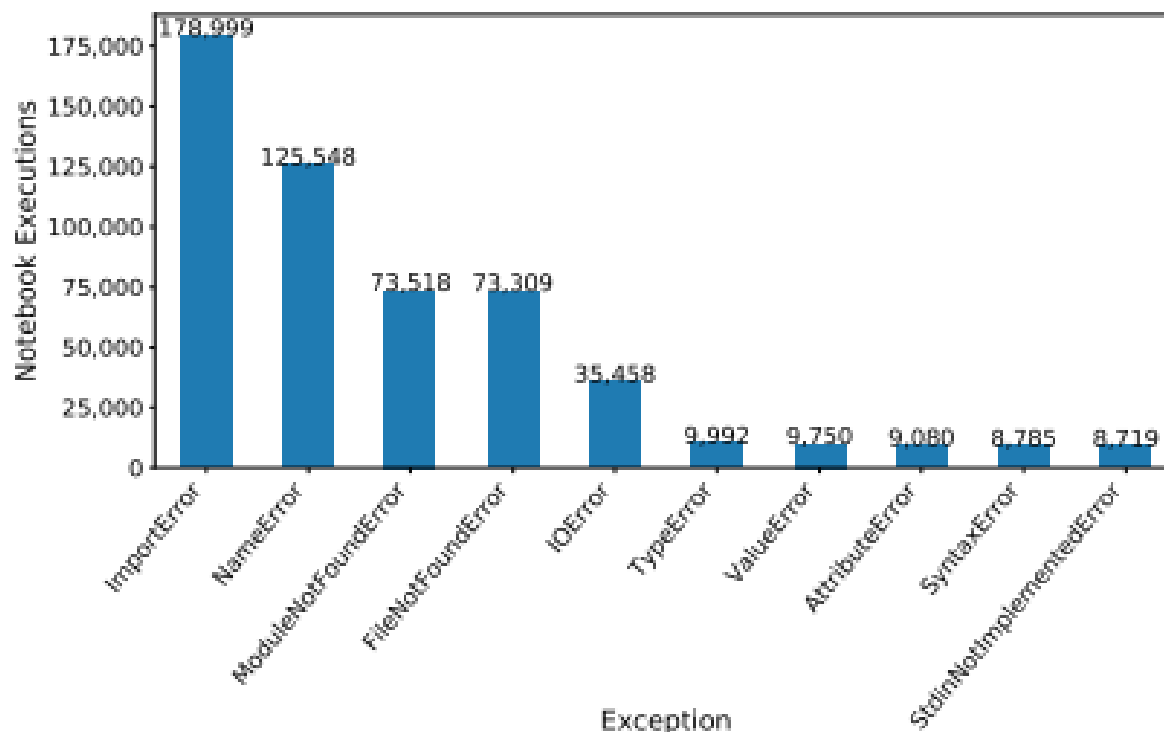


Fig. 12. Top 10 most common exceptions.

# What is another problem?

demo.ipynb		
↑	@@ -47,7 +47,7 @@	
47	47	{
48	48	"data": {
49	49	"text/plain": [
50	-	"<matplotlib.collections.PathCollection at 0x1a16ccdfd0>"
50	+	"<matplotlib.collections.PathCollection at 0x10f636400>"
51	51	]
52	52	},
53	53	"execution_count": 4,
↓	@@ -107,7 +107,7 @@	
107	107	"name": "stdout",
108	108	"output_type": "stream",
109	109	"text": [
110	-	"Value(data=1.1560272899423576, grad=0) 0.5\n"
110	+	"Value(data=0.8958441028683222, grad=0) 0.5\n"
111	111	]
112	112	}
113	113	],
↕	@@ -127,15 +127,15 @@	
127	127	scores = list(map(model, inputs))\n",
128	128	\n",
129	129	# svm \"max-margin\" loss\n",
130	-	losses = [(Value(yi) * scorei + 1).relu() for yi, scorei in zip(yb, scores)]\n",
130	+	losses = [(1 + -yi*scorei).relu() for yi, scorei in zip(yb, scores)]\n",

# And what if I want to run it as a script?

[Papermill](#) or ``python -m jupyter nbconvert --to python XYZ.ipynb` + argparse`

```
$ papermill local/input.ipynb s3://bkt/output.ipynb -p alpha 0.6 -p l1_ratio 0.1
```

But ...

```
#parameters
a = 1
twice = a * 2
```

```
print("a =", a, "and twice =", twice)
```

when executed with `papermill note.ipynb -p a 9`, the output will be `a = 9` and `twice = 2` (not `twice = 18`).

# And what about interactivity?

## IntSlider

```
[2]: widgets.IntSlider(
    value=7,
    min=0,
    max=10,
    step=1,
    description='Test:',
    disabled=False,
    continuous_update=False,
    orientation='horizontal',
    readout=True,
    readout_format='d'
)
```

Test:  4

## Color picker

```
[35]: widgets.ColorPicker(
    concise=False,
    description='Pick a color',
    value='blue',
    disabled=False
)
```

Pick a color



# Exercise: What would be the solution?

- Any suggestions? Ideas?
  - Not using JSON-like format
  - Rerun everything with every change
  - ...

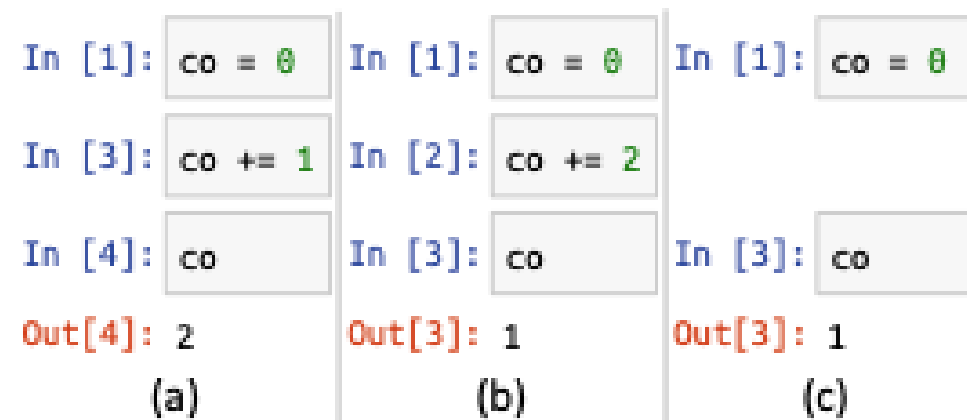


Fig. 2. Three types of Hidden States: (a) Re-execution; (b) edited cell; (c) removed cell.



# Showcase (folder 01)

- Reactivity
  - Python frontend
  - Git-friendly
  - [https://docs.marimo.io/guides/coming\\_from/#5-layouts](https://docs.marimo.io/guides/coming_from/#5-layouts)
- 
- “Decorators” to rescue (see created file)

# How does it try to solve that?

- marimo **statically** analyzes each cell (i.e., without running it) to determine its
  - references, the global variables it reads but doesn't define;
  - definitions, the global variables it defines.
- It then forms a directed acyclic graph (DAG) on cells
- Drawback of static approach:
  - It is not working with complex objects (for example, attributes of the instance)
  - Mutation of the objects (append to the list)
  - → do it in one cell, or do not mutate (create new object)
- [https://docs.marimo.io/guides/best\\_practices/](https://docs.marimo.io/guides/best_practices/)

# How does it try to solve that?

- marimo **static** determine its
  - references,
  - definitions,
- It then forms
- Drawback of
  - It is not work (instance)
  - Mutation of
  - → do it in or
- <https://docs.marimo.io>



out running it) to  
define;  
cells  
attributes of the  
ect)

# AST – Abstract Syntax trees (folder 02)

```
49 def ast_parse(  
50     contents: str, suppress_warnings: bool = True, **kwargs: Any  
51 ) -> ast.Module:  
52     if not suppress_warnings:  
53         return cast(typ=ast.Module, val=ast.parse(source=contents, **kwargs))  
54     with warnings.catch_warnings():  
55         warnings.simplefilter(action="ignore", category=SyntaxWarning)  
56         # The SyntaxWarning is suppressed only inside this `with` block  
57         return cast(typ=ast.Module, val=ast.parse(source=contents, **kwargs))
```

# AST – Abstract Syntax trees

```

49  def ast_p
50      conte
51  ) -> ast.
52      if no
53          r
54      with
55          w
56      #
57      r

```

```

Module(
    body=[
        FunctionDef(
            name='_',
            args=arguments(),
            body=[
                Assign(
                    targets=[
                        Name(id='a', ctx=Store())],
                    value=Constant(value=1)),
                Return(
                    value=Tuple(
                        elts=[
                            Name(id='a', ctx=Load())],
                        ctx=Load()))]),
        FunctionDef(
            name='_',
            args=arguments(
                args=[
                    arg(arg='a')]),
            body=[
                Assign(

```

# AST -> Graphs -> `class ScopedVisitor(ast.NodeVisitor):`

```
✓ class DirectedGraph:      Initial commit, Akshay Agrawal (2 years ago)
    # Nodes in the graph
    cells: dict[CellId_t, CellImpl] = field(default_factory=dict)

    # Edge (u, v) means v is a child of u, i.e., v has a reference
    # to something defined in u
    children: dict[CellId_t, set[CellId_t]] = field(default_factory=dict)

    # Reversed edges (parent pointers) for convenience
    parents: dict[CellId_t, set[CellId_t]] = field(default_factory=dict)
```

## `class ast.NodeVisitor`

A node visitor base class that walks the abstract syntax tree and calls a visitor function for every node found. This function may return a value which is forwarded by the `visit()` method.

This class is meant to be subclassed, with the subclass adding visitor methods.

<https://docs.python.org/3/library/ast.html#ast.NodeVisitor>

```
v: Unknown = ScopedVisitor("cell_" + cell_id)
v.visit(module)
```

Take module (code in the cell) and visit each node in AST ->

```
# ClassDef and FunctionDef nodes don't
> def visit_ClassDef(self, node: ast.Cla
>
> def visit_AsyncFunctionDef(...
>
> def visit_FunctionDef(self, node: ast.
>
> def visit_Call(self, node: ast.Call) -
```

```
nonlocals: set[@Todo] = {name for name in v.defs if not is_local(name)}
temporaries: Unknown = v.defs - nonlocals
variable_data: dict[@Todo, @Todo] = {
    name: v.variable_data[name]
    for name in nonlocals
    if name in v.variable_data
}
```

Get the output for the piece of code

# Now v contains:  
 # - v.defs: set of defined variables  
 # - v.refs: set of referenced variables  
 # - v.variable\_data: metadata about each variable

```
return CellImpl(
    # keyed by original (user) code, for cache lookups
    key=code_key(code=code),
    code=code,
    mod=original_module,
    defs=nonlocals,
    refs=v.refs,
    sql_refs=v.sql_refs,
    temporaries=temporaries,
    variable_data=variable_data,
```

Get the internal representation of input/output – with references

body = bytecode



# So we have a graph. Now what? Need to show it

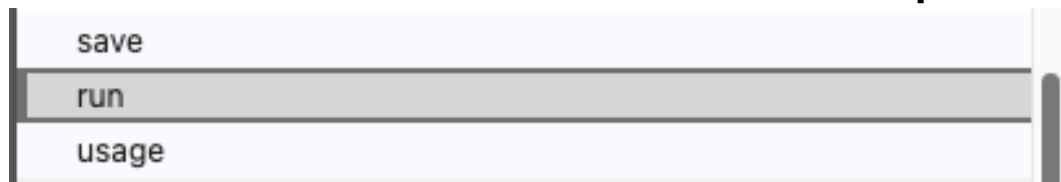
- Exercise number 2 😊
- Options? What do you think?
  - Infinite loop



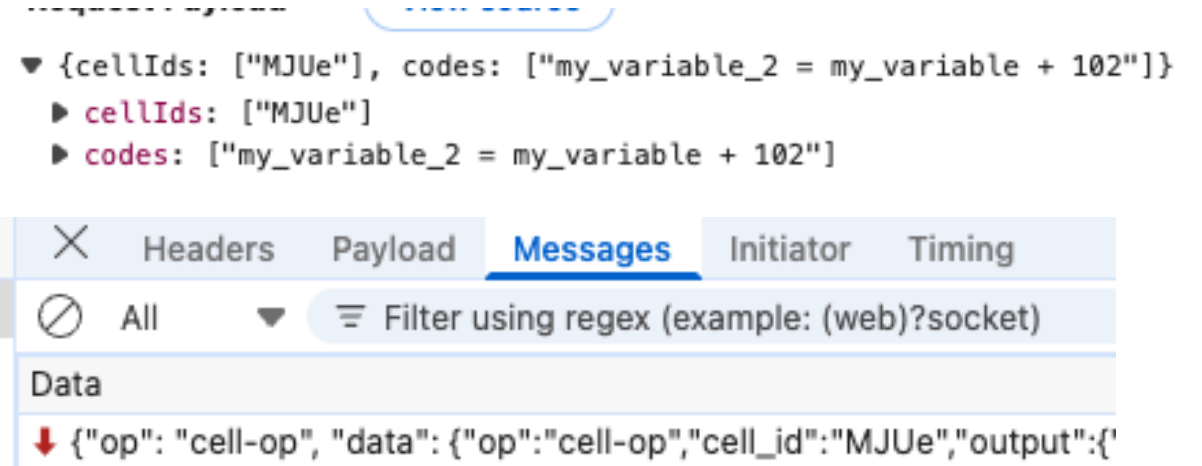


# How to create responsive UI? -> Websockets

- Websocket (on top of <https://starlette.dev/> - light ASGI used by FastAPI)
- Frontend listen for updates on each cell
- # Frontend send a change to backend (to the queue)
- # Backend will execute cell (with a Graphs of dependency in mind)
- # Backend will send an update to frontend



Name
ws?file=marimo_showcase.py&session_id=s_cri8s7



# Backend

```
@router.websocket("/ws")
async def websocket_endpoint(websocket: WebSocket) -> None:
    # 1. Authentication check
    if app_state.enable_auth and not validate_auth(websocket):
        await websocket.close(WebSocketCodes.UNAUTHORIZED, "MARIMO_UNAUTHORIZED")
        return

    # 2. Extract query parameters
    session_id = SessionId(raw_session_id)
    file_key = app_state.query_params(FILE_QUERY_PARAM_KEY)
    kiosk = app_state.query_params(KIOSK_QUERY_PARAM_KEY) == "true"

    # 3. Create WebSocketHandler
    await WebSocketHandler(
        websocket=websocket,
        manager=app_state.session_manager,
        session_id=session_id,
        file_key=file_key,
        kiosk=kiosk,
        ...
    ).start()
```

```
def write_operation(self, op: MessageOperation) -> None:
    self.message_queue.put_nowait(serialize_kernel_message(op))
```

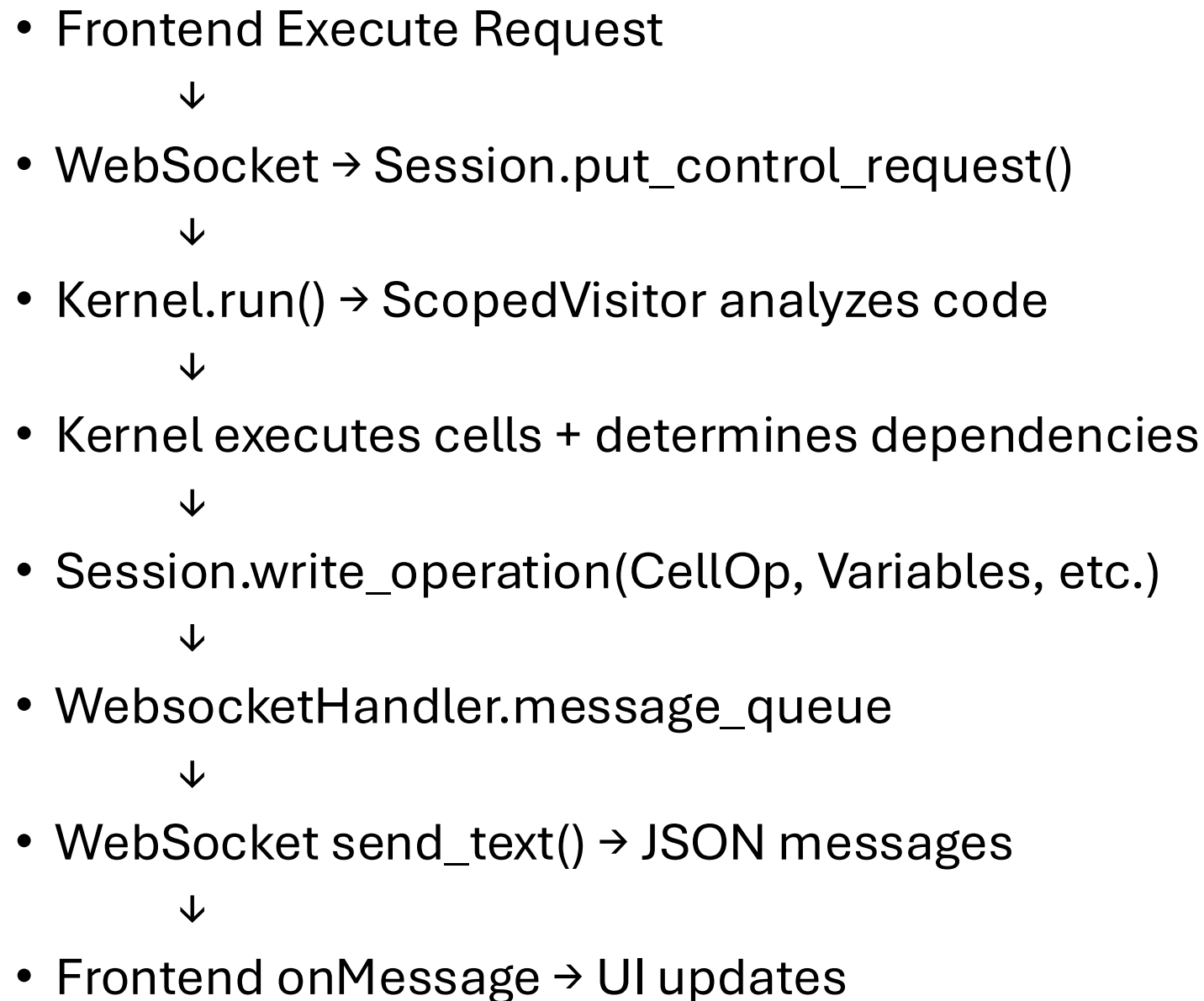
```
v: Unknown = ScopedVisitor("cell_" + cell_id)
v.visit(module)
```

# Frontend

```
// In useWebSocket.tsx
const socket: IReconnectingWebSocket =
    isWasm() ? new PyodideWebSocket(PyodideBridge.INSTANCE)
    : options.static ? new StaticWebSocket()
    : new ReconnectingWebSocket(rest.url, undefined, {
        maxRetries: 10,
        startClosed: true,
        connectionTimeout: 10_000,
    });
```

```
onMessage: (event: WebSocketEventMap["message"]) => {
    const message = JSON.parse(event.data);

    // Route to appropriate handler based on message.op
    switch (message.op) {
        case "kernel-ready":
            // Initialize cells and UI state
            break;
        case "cell-op":
            // Update cell output
            break;
        case "variables":
            // Update variable values
            break;
        // ... many more operation types
    }
}
```



# Export to HTML (folder 03)

- There is no server?? How it can work?

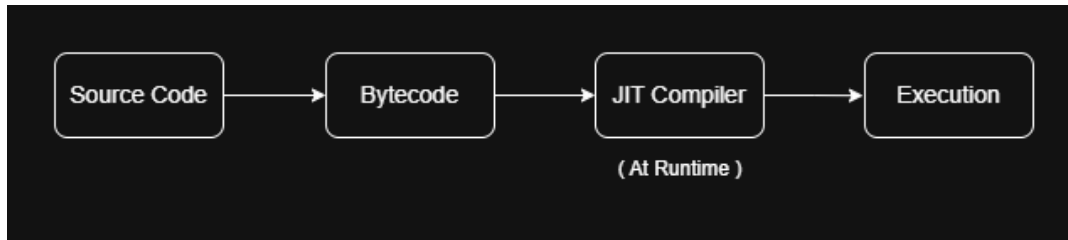
# WebAssembly



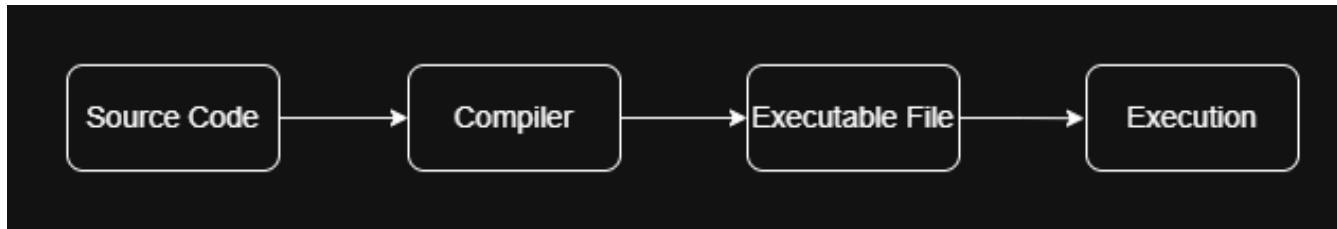
# We have 3 ways how to run the code

It is not that simple even for Python ... but let's ignore it for now

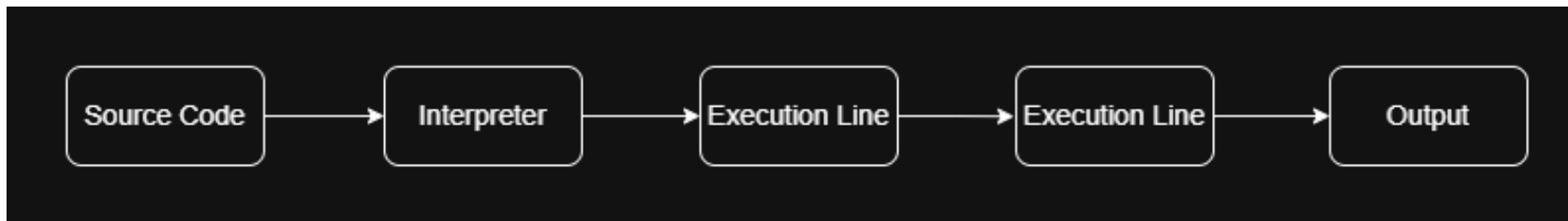
- JavaScript (V8) – Just-in-time



- WASM – Ahead-of-time compilation



- Python - CPython – Interpreter-based



C source code and corresponding WebAssembly

C source code	WebAssembly .wat text format	WebAssembly .wasm binary format
<pre>int factorial(int n) {     if (n == 0) {         return 1;     } else {         return n *         factorial(n - 1);     } }</pre>	<pre>(func (param i64)   (result i64)     local.get 0     i64.eqz     if (result i64)       i64.const 1     else       local.get 0       local.get 0       i64.const 1       i64.sub       call 0       i64.mul     end)</pre>	<pre>00 61 73 6D 01 00 00 00 01 06 01 60 01 7E 01 7E 03 02 01 00 0A 17 01 15 00 20 00 50 04 7E 42 01 05 20 00 20 00 42 01 7D 10 00 7E 0B 0B</pre>

# WebAssembly 101

- WebAssembly is a new type of code that can be run in modern web browsers and provides new features and major gains in performance. **It is not primarily intended to be written by hand**, rather it is designed to be an effective **compilation target** for source languages like C, C++, Rust, etc.
- WebAssembly is a different language from JavaScript, but it is not intended as a replacement.
- The virtual machine in the browser can now load and run two types of code — JavaScript AND WebAssembly.
- WebAssembly modules can be loadable just like ES modules (using `<script type='module'>`), meaning that JavaScript will be able to fetch, compile, and import a WebAssembly module as easily as an ES module.

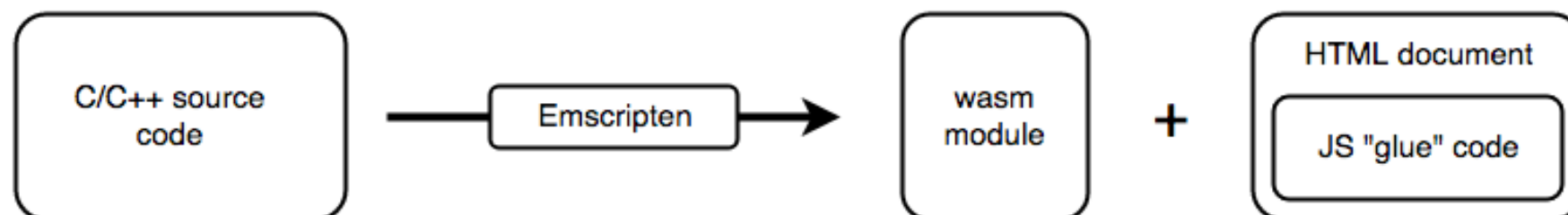
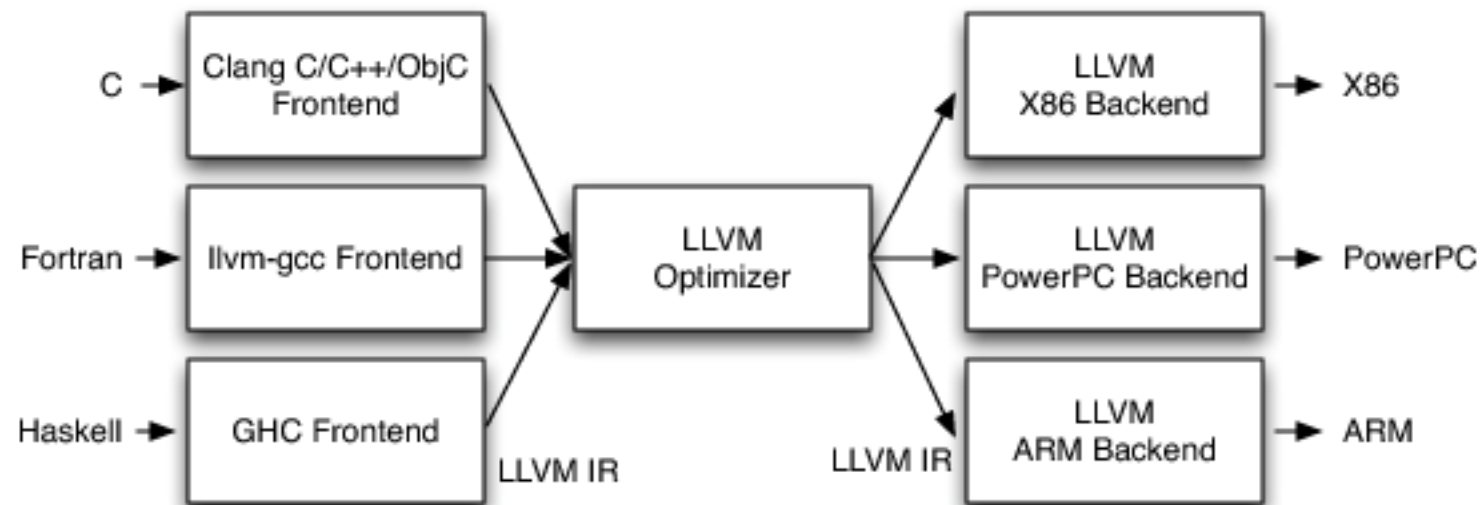


# How to get WASM code?

- **Porting a C/C++ application with [Emscripten](#).**
- Writing or generating WebAssembly directly at the assembly level.
- Writing a Rust application and targeting WebAssembly as its output.
- Using [AssemblyScript](#) which looks similar to TypeScript and compiles to WebAssembly binary.

# Emscripten

- Emscripten is a complete compiler toolchain to WebAssembly, using **LLVM**, with a special focus on speed, size, and the Web platform.



# Where to run this WASM code? (folder 04)

- “Server” side:
  - [Wasmtime](#) (even [Python](#) version)
  - [WasmEdge](#)
  - Lucet
- “Client + server” side:
  - Javascript (v8)

<https://v8.dev/docs/wasm-compilation-pipeline>

[https://developer.mozilla.org/en-US/docs/WebAssembly/Guides/Loading\\_and\\_running\\_a\\_Wasm\\_module\\_in\\_a\\_browser](https://developer.mozilla.org/en-US/docs/WebAssembly/Guides/Loading_and_running_a_Wasm_module_in_a_browser)

<https://nodejs.org/en/learn/getting-started/nodejs-with-webassembly>

<https://dev.to/begoon/compile-c-to-webassembly-wasm-and-run-it-in-the-browser-h1l>

```

1 // Assume add.wasm file exists that contains a single function
2 const fs = require('node:fs');
3
4 // Use the readFileSync function to read the contents of the "a
5 const wasmBuffer = fs.readFileSync('/path/to/add.wasm');
6
7 // Use the WebAssembly.instantiate method to instantiate the We
8 WebAssembly.instantiate(wasmBuffer).then(wasmModule => {
9     // Exported function lives under instance.exports object
10     const { add } = wasmModule.instance.exports;
11     const sum = add(5, 6);
12     console.log(sum); // Outputs: 11
13 });
    
```

# WASM, so what?

# WASM, so what?



# WebAssembly + Python?

CPython is the key

# It is getting some attention in PEP

- <https://peps.python.org/pep-0776/> (PEP 776 – Emscripten Support)
- <https://peps.python.org/pep-0783/> (PEP 783 – Emscripten Packaging)
- Similar way is to use Python-to-C mapping -> WASM
  - <https://github.com/wasmerio/py2wasm> using <https://nuitka.net/>
  - Nuitka translates the Python modules into a C level program that then uses libpython and static C files of its own to execute in the same way as CPython does.

# Pyodide

- Pyodide is a port of CPython to WebAssembly/[Emscripten](#).
- micropip (not confuse with microPython 😊)
- <https://pyodide.org/en/stable/console.html>
- <https://stanford.edu/~cpiech/bio/papers/pyodideU.pdf>



# Other ways of using Python in browser

- <https://pyscript.net/>
- <https://pypyjs.org/>

# Return from rabbit hole -> Marimo + WASM

- Folder 05 – practical examples
- <https://pyvideo.org/pycon-us-2025/marimo-a-notebook-that-compiles-python-for-reproducibility-and-reusability.html>