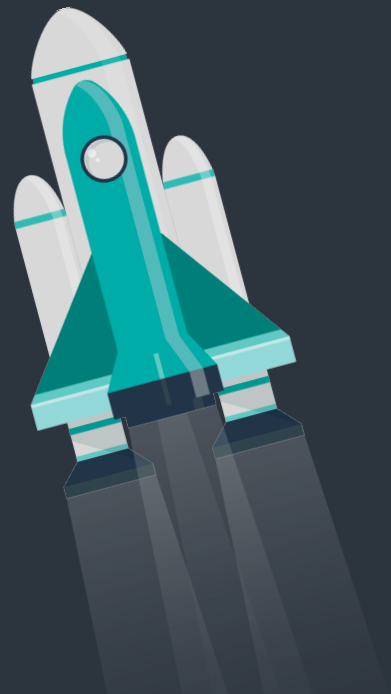
Runtime Scripting for Rust Applications

Niklas Korz

About Me

- Co-founder and tech lead at **alugha.com**
- Meetup organizer: “Nix Your Bugs & Rust Your Engines”
- Likes:
  - distributed and reproducible systems
  - natural languages
  - pen & paper roleplaying games
- Website: **https://korz.dev**
- Mastodon: **@niklaskorz@rheinneckar.social**

1. Motivation
2. Languages and Implementations
3. Embedding Deno
4. Further Reading



Rust is…

- statically typed
- compiled ahead of time
- borrow checked

When to Script

- Fast prototyping
  - Hot reloading
- Allow end-users to change runtime behavior
  - C-ABI? **Stable Rust ABI**?
  - Sandboxed execution
- Collaborate with people who find Rust too complicated

Rust is…

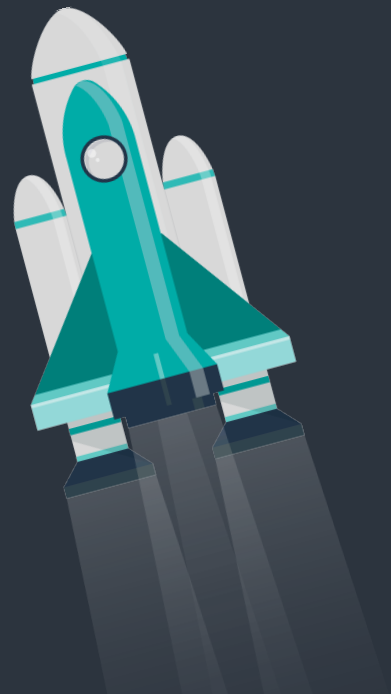
- statically typed
- compiled ahead of time
- borrow checked

Python* is…

- dynamically/gradually typed 
- interpreted (or JIT compiled)
- garbage collected

* and JavaScript, Lua, Ruby, …

1. Motivation
2. Languages and Implementations
3. Embedding Deno
4. Further Reading



WebAssembly

- Portable binary code, usually JIT-compiled
- Stable call interface (more **powerful ABIs** are proposed)
- Sandboxed execution
- Cross-language
  - but you have to bring your own allocator

    (**unless you're using garbage collection**)

Built for Rust

- **Rhai**:
  - AST-interpreter (relatively slow)
  - Dynamically typed
  - Custom operators
- **Mun**:
  - AOT compiled, statically typed
  - Designed for hot reloading
  - LLVM for compilation and optimization

https://github.com/mun-lang/mun/issues/418

Open

String type #418

aghoneim92 opened this issue on Jul 7, 2022 · 6 comments

trsh commented on Jul 24

Is there still no string support?

Wodann commented on Jul 30 — Collaborator

> Is there still no string support?

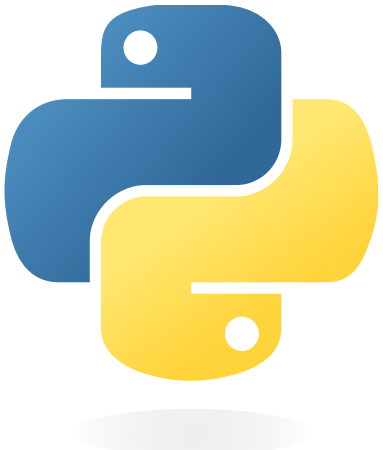
No. We currently have higher priority tasks on our roadmap, so it's unlikely that strings will land in the near future.

😕 1

Lua

- Fast: stack-based interpreter or JIT compiled
- Lightweight (code: 355kb **Lua** vs 25mb **CPython** vs 37mb **v8**)
- **mlua** for Rust:
  - wraps C Lua implementations (Lua/LuaJIT/Luau)
  - async/await support
  - sandboxing (with Luau)

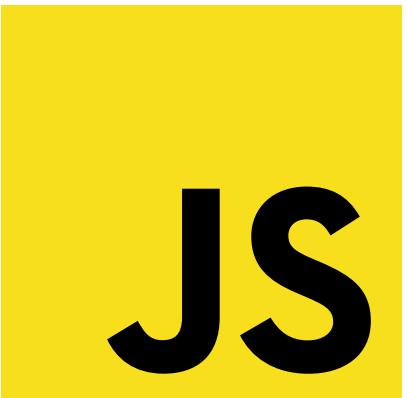
Python



- Bytecode interpreter (at least in CPython)
- optional static typing
- **PyO3**: wraps CPython, often used for libraries
- **RustPython**: pure Rust implementation, (yet) incomplete
- Both support async/await

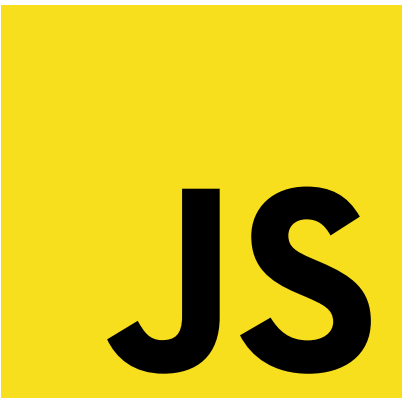
ECMAScript*

- Standardized (**ECMA-262**)
- Optional static typing (**almost standard**)
- Fast: mix of interpreter and JIT
- Multiple major implementations:
  - Gecko (Firefox): **SpiderMonkey**
  - WebKit (Safari): **JavaScriptCore**
  - Blink (Chrome): **v8**



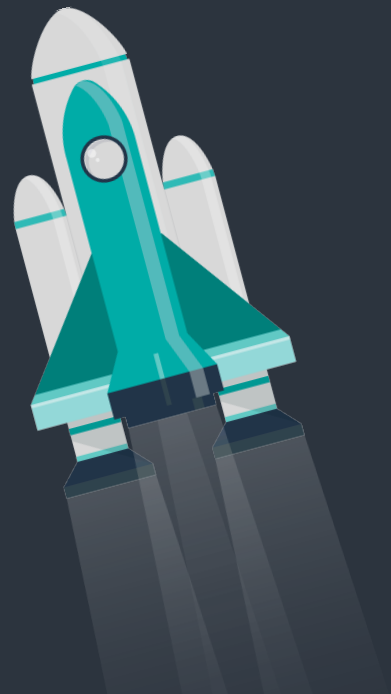
* or JavaScript or TypeScript

ECMAScript in Rust



- **Deno**: v8-based Node.js competitor
  - mature and stable
  - implements many Web APIs like fetch, localStorage
- **rusty_jsc**: based on JavaScriptCore
  - relatively young, easy to use
- **mozjs**: based on SpiderMonkey, used in **Servo**
  - complicated API, requires use of `unsafe`

1. Motivation
2. Languages and Implementations
3. Embedding Deno
4. Further Reading



Embedding Deno in Rust

- **deno_core**:
  - wraps v8 in a safe Rust API (**rusty_v8**)
  - provides extension mechanism “ops”
  - implements serde for v8 objects (beware of UTF-16)
  - import hooks: `ModuleLoader`

Embedding Deno in Rust

- **deno_ast**:
  - transpiler for TypeScript, JSX and ES proposals
  - based on **SWC** (Rust alternative to Babel)
  - extensible: SWC plugins in WebAssembly or ES

Embedding Deno in Rust

- **deno_runtime**:
  - batteries included
  - implements Web APIs such as:
    - fetch, localStorage, WebGPU, Web Workers, Web Sockets
  - additional APIs:
    - file system access, HTTP server, TCP/UDP sockets

Communicating between Rust and JS

- Deno Extensions: provide native “ops” and some JS glue
  - ops only accessible inside the JS glue code
  - `import { op_name_here } from "ext:core/ops"`
  - assign to `globalThis` to expose
- serde_v8: converts between JS types and Rust types

```rust
struct HostState {
    pub n: i32,
}

let host_state = Arc::new(RwLock::new(
  HostState { n: 42 },
));

let mut state = worker.js_runtime.op_state().borrow_mut();
state.put(host_state);
```

```rust
#[op2(async)]
pub async fn op_scripting_demo(
  state: Rc<RefCell<OpState>>,
  n: i32,
) -> i32 {
    let lock = state.borrow()
      .borrow::<Arc<RwLock<HostState>>>().clone();
    let mut host_state = lock.write().await;
    host_state.n += n;
    host_state.n
}
```

```
extension!(
    my_extension,
    ops = [op_scripting_demo],
    esm_entry_point = "ext:my_extension/bootstrap.js",
    esm = ["bootstrap.js"],
    docs = "A small sample extension"
);
```

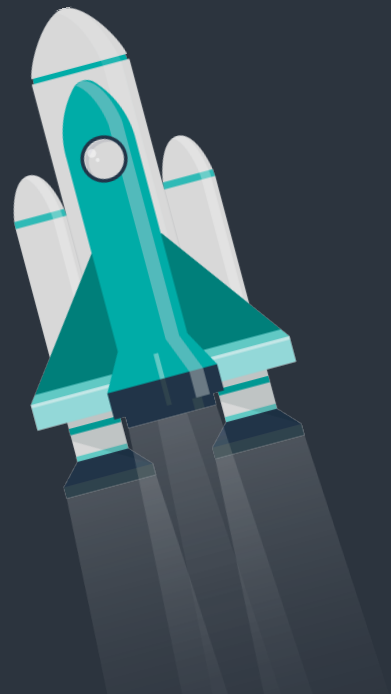
```
// bootstrap.js

import { op_scripting_demo } from "ext:core/ops";

globalThis.opScriptingDemo = op_scripting_demo;
```

```ts
// builtins/state.ts

export function addToHostState(
  n: number,
): Promise<number> {
  return globalThis.opScriptingDemo(n);
}
```

```ts
// example.ts
import { addToHostState } from "builtin:state";

export default async function demo() {
  const response = await fetch("https://api.ipify.org");
  const text = await response.text();
  console.log("IP:", text);

  console.log("State:", await addToHostState(2));
  console.log("State:", await addToHostState(3));
}
```

```
$ cargo run -- example.ts
...

IP: 13.248.252.114
State: 44
State: 47
```

1. Motivation
2. Languages and Implementations
3. Embedding Deno
4. Further Reading



Further Reading

- Example: **https://github.com/niklaskorz/eurorust2024**
- Deno: **Roll your own JavaScript runtime** (three parts)
- Deno: **Announcing Stable V8 Bindings for Rust**
- Deno: **The Internals of Deno** by Mayank Choubey
- PyO3: **Calling Python from Rust**
- mlua: **bevy_mod_scripting** by Maksymilian Mozolewski
- mlua: **Lua in the Browser, with Rust and WebAssembly** by Ruben Otto

Questions?

- Download: **https://dl.korz.dev/eurorust2024.pdf**
- Contact me on…
  - Email: **contact@korz.dev**
  - Mastodon: **@niklaskorz@rheinneckar.social**
  - Matrix