RustPython

FOSDEM 2019

Brought to you by: Shing and Windel :P

Outline

- Who are we?
- What is python? What is rust? What is the problem with C?
- Overview of RustPython internals parser/compiler/vm/imports
- Commandline demo
- WebAssembly
- WebAssembly demo

Whoami

Windel Bouwman

Software engineer at Demcon

Python and open source fan

Main author of ppci -> check this out!

https://github.com/windelbouwman/

Twitter: @windelbouwman

Whoami

Shing Lyu

Software Engineer at DAZN

ex-Mozilla employee (Servo, Quantum, Firefox)

https://github.com/shinglyu

https://shinglyu.github.io/blog/

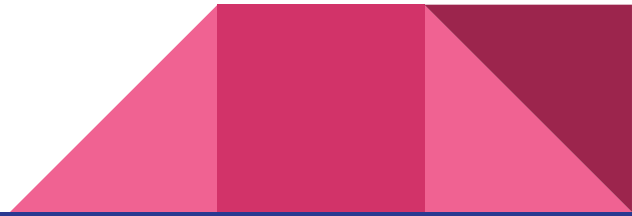
RustPython project

What is RustPython?

- A Python implementation in Rust
- Python 3+ syntax
- Homepage: https://github.com/RustPython/RustPython
- Community
  - 19 contributors, about 5 larger contributors
- Project status:
  - Early phase, most syntax works
  - WebAssembly demo working
  - Not much standard library

Why did we do this?

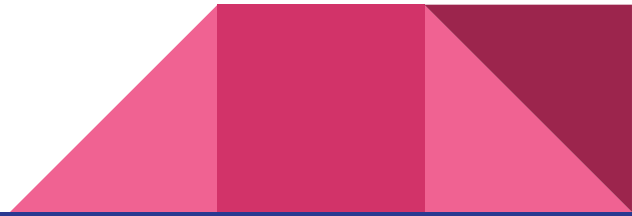
- Rust is a safer language then C
  - In general: Rust allows you to focus on actual application
- Learn rust
- Learn python internals
- Create a new Python implementation which is more memory safe



RustPython internals

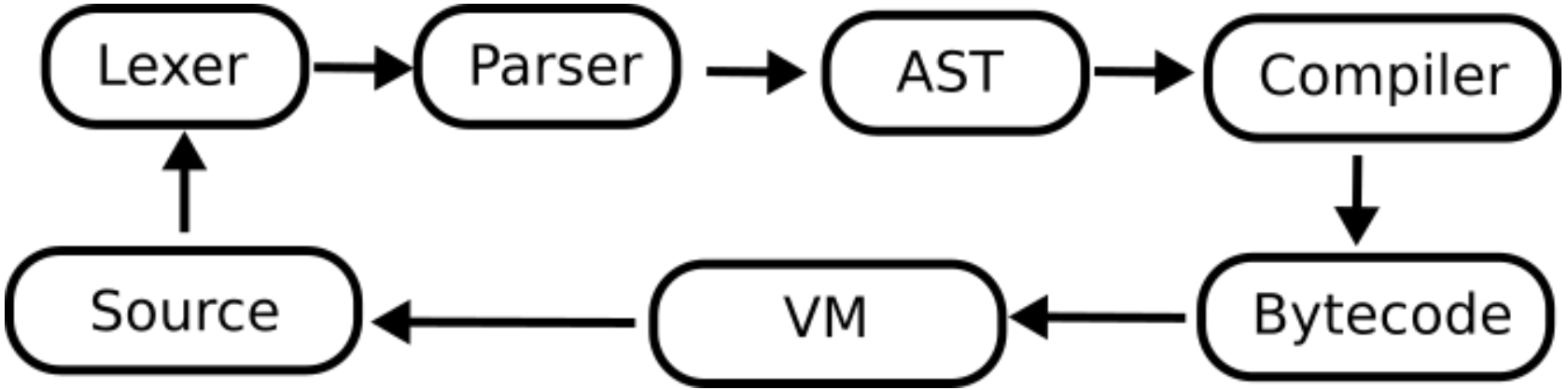
RustPython internals

- Rust Crates
- Lexer, parser, AST (Abstract Syntax Tree)
- Compiler
- VM (Virtual Machine)
- Import system
- Builtin objects



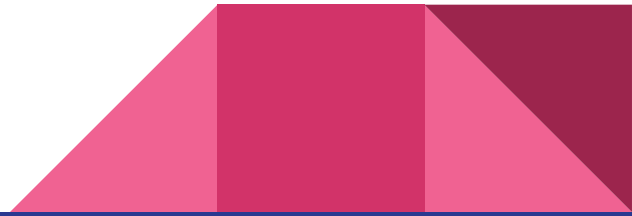
Overall design

Follow CPython strategy



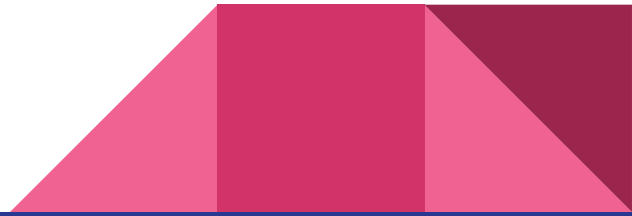
Rust Crates

- rustpython_parser: The lexer, parser and AST
- rustpython_vm: The VM, compiler and builtin functions
- rustpython: Using the above crates to create an interactive shell



Lexing, parsing, AST

- A manual written lexer to deal with indent and dedent of Python
  - Task: Convert Python source into tokens
- The parser is generated with LALRPOP (https://github.com/lalrpop/lalrpop)
  - Task: Convert tokens into an AST
- The AST (abstract syntax tree) nodes are Rust structs and enums



Compiler and bytecode

- The compiler turns python syntax into bytecode
- CPython bytecode is not stable and varies wildly between versions.
- Example bytecode →
- Idea: standardize this bytecode between Python implementations?

```
import dis
def f(a, b):
    return a + b + 3
dis.dis(f.__code__)
```

```
  3           0 LOAD_FAST                0 (a)
              2 LOAD_FAST                1 (b)
              4 BINARY_ADD
              6 LOAD_CONST               1 (3)
              8 BINARY_ADD
             10 RETURN_VALUE
```

Virtual Machine (VM)

- A fetch and dispatch loop

```
match &instruction {
    bytecode::Instruction::LoadConst { ref value } => {
        let obj = self.unwrap_constant(vm, value);
        self.push_value(obj);
        Ok(None)
    }
    bytecode::Instruction::Import {
        ref name,
        ref symbol,
    } => self.import(vm, name, symbol),
```

Object model

- Use Rust Rc and RefCell to do reference counting of Python objects
- Optionally store rust payload (for instance String, or f64)

```rust
pub type PyRef<T> = Rc<RefCell<T>>;
pub type PyObjectRef = PyRef<PyObject>;

pub struct PyObject {
    pub kind: PyObjectKind,
    pub typ: Option<PyObjectRef>,
    pub dict: HashMap<String, PyObjectRef>,
}
```

```rust
pub enum PyObjectKind {
    String {
        value: String,
    },
    Integer {
        value: BigInt,
    },
    Float {
        value: f64,
    },
    Complex {
        value: Complex64,
    },
    Bytes {
        value: Vec<u8>,
    },
```

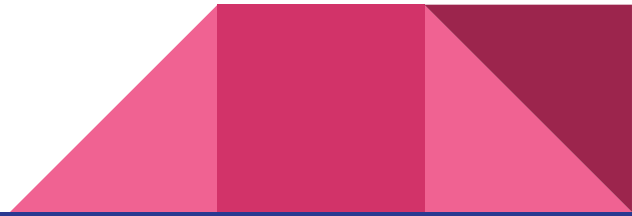
Builtin functions

- Builtin Python functions are implemented in Rust like this

```
fn builtin_all(vm: &mut VirtualMachine, args: PyFuncArgs) -> PyResult {
    arg_check!(vm, args, required = [(iterable, None)]);
    let items = vm.extract_elements(iterable)?;
    for item in items {
        let result = objbool::boolval(vm, item)?;
        if !result {
            return Ok(vm.new_bool(false));
        }
    }
    Ok(vm.new_bool(true))
}
```

Demo time!

- Run rustpython from commandline now!
- Git clone https://github.com/RustPython/RustPython
- cargo run

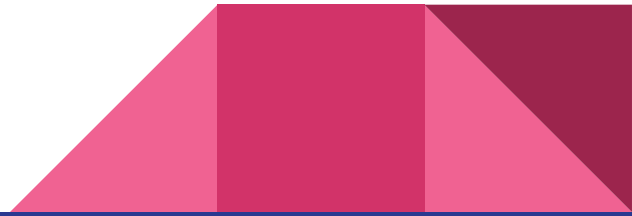


Notable current challenges

- Ask for your help (since this is the rust devroom :D)
- The python dict
- The standard library

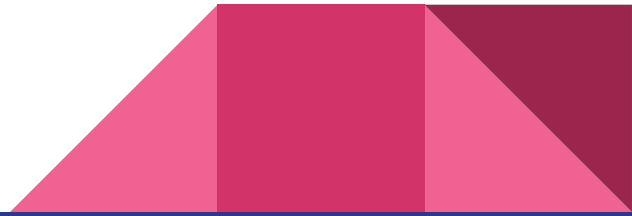
Challenge: the Python dict

- Rust has a HashMap type
- To implement Python the dict type, HashMap is tempting, but…
- Every python object can be a dict key, if it implements __hash__ and __eq__.
- Both these methods can raise an exception…
- HashMap does not permit for failing hashes…
- Now what? Own hash map implementation? :(



Challenge: the standard library

- A lot of the Python standard library is written in Python and can be shared between implementations
- See also: Ouroboros (https://github.com/pybee/ouroboros)
- How to not duplicate code too much between Python implementations?

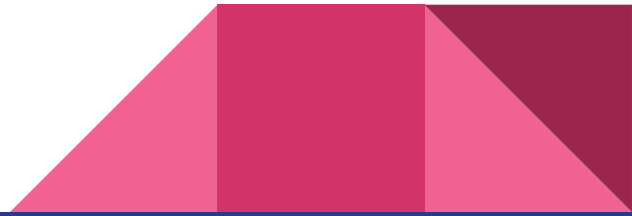


WebAssembly

What is WebAssembly?

- Low-level assembly-like language (+ binary format)
- Runs with near-native performance in browsers
- Work together with JavaScript
- As a Rust compile target

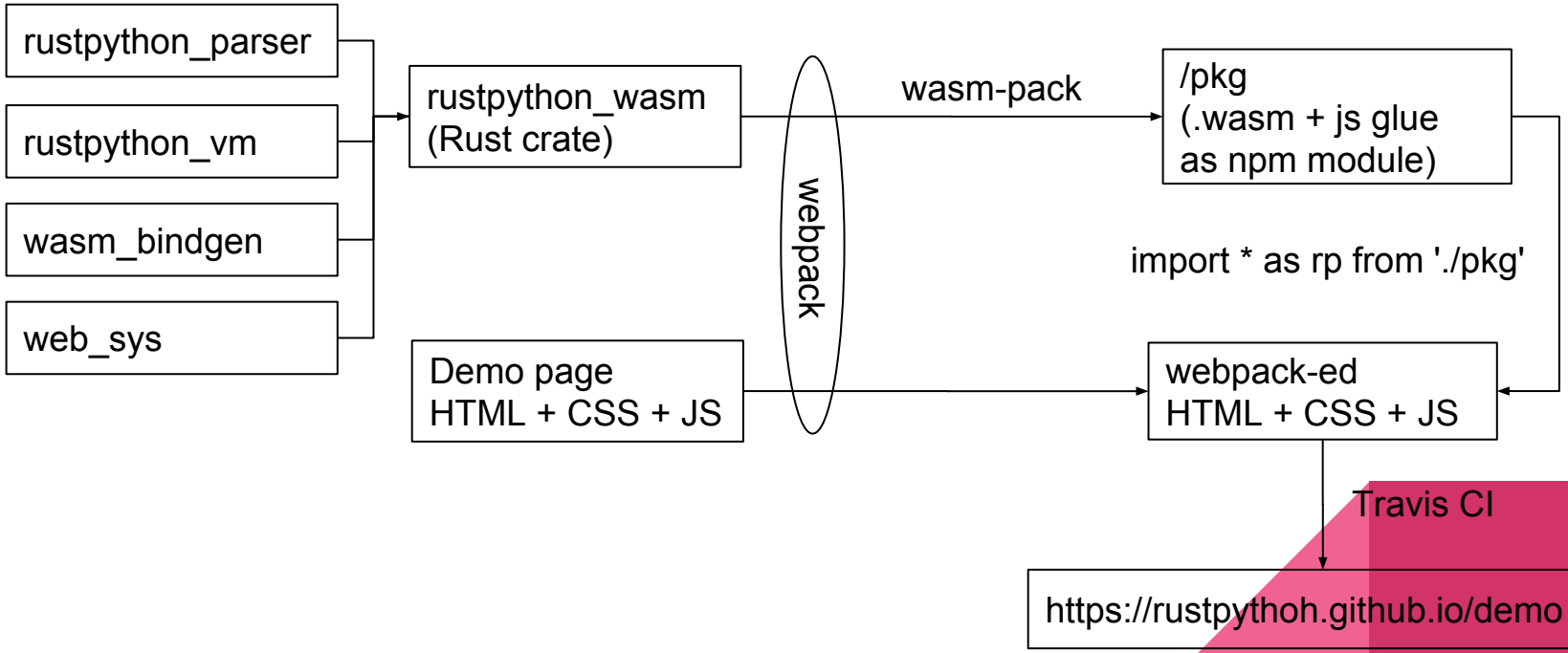
- Big shout-out to Ryan Liddle (rmliddle) for porting RustPyhon to WASM



The toolchain

- wasm-pack
  - wasm-bindgen
  - web_sys
  - webpack + wasm-pack-plugin
- Travis CI
- gh-pages

WASM Workflow



Exposing the eval() to JavaScript

```
#[wasm_bindgen(js_name = pyEval)]
pub fn eval_py(source: &str, options: Option<Object>) -> Result<JsValue, JsValue> {
    // Setting up the VirtualMachine and stuff
    eval(&mut vm, source, vars)
        .map(|value| py_to_js(&mut vm, value))
        .map_err(|err| py_str_err(&mut vm, &err).into())
}
```

Using eval() in JavaScript

```
// The rustpython_wasm Rust crate is in ../../lib/
import * as rp from '../../lib/pkg';

code = 'print(42)'

const result = rp.pyEval(code, {
    stdout: '#console'
});
```

Python print() to JS console.log()

```rust
use web_sys::{console};

pub fn builtin_print_console(vm: &mut VirtualMachine, args: PyFuncArgs) -> PyResult {
    let arr = Array::new();
    for arg in args.args {
        arr.push(&vm.to_pystr(&arg)?.into());
    }
    console::log(&arr);
    Ok(vm.get_none())
}
```

Python print() to an HTML <textarea>

```rust
use web_sys::{window, HtmlTextAreaElement};

pub fn print_to_html(text: &str, selector: &str) -> Result<(), JsValue> {
    let document = window().unwrap().document().unwrap();
    let element = document
        .query_selector(selector)?
        .ok_or_else(|| js_sys::TypeError::new("Couldn't get element"))?;
    let textarea = element
        .dyn_ref::<HtmlTextAreaElement>()
        .ok_or_else(|| js_sys::TypeError::new("Element must be a textarea"))?;
    let value = textarea.value();
    textarea.set_value(&format!("{}{}", value, text));
    Ok(())
}
```

Web-based demo

https://rustpython.github.io/demo/

Future steps?

- A JavaScript replacement for client-side scripting? (check Brython)
- Python IDE in browser?
- Pure client-side Jupyter Notebook (IPython Notebook)? (check Iodide)
- Data science, AI?

```python
# Brython Example code
from browser import document, html

element = document.getElementById("zone6_std")
nb = 0

def change(event):
    global nb
    elt = document.createElement("B")
    txt = document.createTextNode(f" {nb}")
    elt.appendChild(txt)
    element.appendChild(elt)
    nb += 1

document["button6_std"].addEventListener("click", change)
```

Questions?

- Thank you for your attention!
- https://github.com/RustPython/RustPython
- https://github.com/windelbouwman/
- https://github.com/shinglyu/

Backup

$ ppci-wabt show_interface rust_python.wasm

- This interface is better than the emscripten compiled micropython/cpython -> much less dependencies on libc.

Imports:

./rustpython_wasm.__wbindgen_string_new: [i32, i32] -> [i32]

./rustpython_wasm.__wbindgen_object_drop_ref: [i32] -> []

./rustpython_wasm.__widl_instanceof_Window: [i32] -> [i32]

./rustpython_wasm.__widl_f_get_element_by_id_Document: [i32, i32, i32] -> [i32]

./rustpython_wasm.__widl_instanceof_HTMLTextAreaElement: [i32] -> [i32]

./rustpython_wasm.__widl_f_value_HTMLTextAreaElement: [i32, i32] -> []

./rustpython_wasm.__widl_f_set_value_HTMLTextAreaElement: [i32, i32, i32] -> []

./rustpython_wasm.__widl_f_document_Window: [i32] -> [i32]