DS-210: PROGRAMMING FOR DATA SCIENCE

LECTURE 38

CALLING RUST FROM PYTHON

OFFICE HOURS THIS WEEK

- Today: 2:30–3:30
- Wednesday: 4:30–6 (will finish slightly early)

OFFICE HOURS THIS WEEK

- Today: 2:30–3:30
- Wednesday: 4:30–6 (will finish slightly early)

- Proposed final project coding jams:
    - Wednesday: 7–???
    - Thursday: 8–???
    - (and another one on Monday?)

TWO MAIN CHALLENGES

1. Make Rust and Python talk to each other
    - create a dynamic library in Rust
    - make Python call it

TWO MAIN CHALLENGES

1. Make Rust and Python talk to each other
    - create a dynamic library in Rust
    - make Python call it

2. Make them speak the same language: **foreign function interface**

    - data layout
    - how the parameters are passed to the function
    - common language: **C**

CREATE A DYNAMIC LIBRARY IN RUST

Library types:

- Dynamic library (separate file)
    - call it when needed
- Static library (part of the executable)
    - included during compilation

Resulting *dynamic* library file:

- Windows: extension .dll
- Linux: extension .so

- Specifics operating system dependent

CREATE A DYNAMIC LIBRARY IN RUST

Library types:

- Dynamic library (separate file)
    - call it when needed
- Static library (part of the executable)
    - included during compilation

Resulting *dynamic* library file:

- Windows: extension .dll
- Linux: extension .so
- Specifics operating system dependent

1. Create a new project: `cargo new --lib your_library_name`
2. Add section in `Cargo.toml`:

```
[lib]
name = "your_library_name"
crate-type = ["dylib"]
```

CREATE A DYNAMIC LIBRARY IN RUST

Library types:

- Dynamic library (separate file)
  - call it when needed
- Static library (part of the executable)
  - included during compilation

Resulting *dynamic* library file:

- Windows: extension .dll
- Linux: extension .so
- Specifics operating system dependent

1. Create a new project: `cargo new --lib your_library_name`
2. Add section in `Cargo.toml`:

```
[lib]
name = "your_library_name"
crate-type = ["dylib"]
```

- Running `cargo build` will create `libyour_library_name.so` (Linux)

File Edit View Search Terminal Help

```
[user@lecture-38 ds210]$ mkdir addition
[user@lecture-38 ds210]$ cd addition/
[user@lecture-38 addition]$
```

File Edit View Search Terminal Help

```
[user@lecture-38 ds210]$ mkdir addition
[user@lecture-38 ds210]$ cd addition/
[user@lecture-38 addition]$ cargo new --lib rust_functions
     Created library `rust_functions` package
[user@lecture-38 addition]$
```

```
[package]
name = "rust_functions"
version = "0.1.0"
edition = "2021"

# See more keys and their definitions at https://doc.rust-lang.org/cargo/reference/manifest.html

[dependencies]

[lib]
name = "rust_functions"
crate_type = ["dylib"]
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
```

File Edit View Search Terminal Help

```
[user@lecture-38 addition]$ cd rust_functions/
[user@lecture-38 rust_functions]$ cargo build --release
   Compiling rust_functions v0.1.0 (/home/user/ds210/addition/rust_functions)
    Finished release [optimized] target(s) in 0.23s
[user@lecture-38 rust_functions]$
```

File Edit View Search Terminal Help

```
[user@lecture-38 addition]$ cd rust_functions/
[user@lecture-38 rust_functions]$ cargo build --release
   Compiling rust_functions v0.1.0 (/home/user/ds210/addition/rust_functions)
    Finished release [optimized] target(s) in 0.23s
[user@lecture-38 rust_functions]$ tree
.
├── Cargo.lock
├── Cargo.toml
├── src
│   └── lib.rs
└── target
    ├── CACHEDIR.TAG
    └── release
        ├── build
        ├── deps
        │   ├── librust_functions.so
        │   └── rust_functions.d
        ├── examples
        ├── incremental
        ├── librust_functions.d
        └── librust_functions.so

7 directories, 8 files
[user@lecture-38 rust_functions]$
```

FORMAT OF FUNCTIONS IN THE LIBRARY (`lib.rs`)

Specify that:

- C conventions should be used (`extern "C"`)
- name should not be modified (`#[no_mangle]`)

```
#[no_mangle]
pub extern "C" fn add(x: i32, y: i32) -> i32 {
    x + y
}
```

```rust
#[cfg(test)]
mod tests {
    #[test]
    fn it_works() {
        let result = 2 + 2;
        assert_eq!(result, 4);
    }
}
```

File Edit View Search Terminal Help

```
#[no_mangle]
pub extern "C" fn add(x: i32, y: i32) -> i32 {
    x + y
}
```

~

"src/lib.rs" 4L, 72B 4,1 All

File Edit View Search Terminal Help

```
[user@lecture-38 rust_functions]$ cargo build --release
   Compiling rust_functions v0.1.0 (/home/user/ds210/addition/rust_functions)
    Finished release [optimized] target(s) in 0.29s
[user@lecture-38 rust_functions]$ cd ..
[user@lecture-38 addition]$
```

USING FOREIGN FUNCTION INTERFACE IN PYTHON

```
from cffi import FFI

ffi = FFI()
ffi.cdef("""
    int add(int, int);
""")

C = ffi.dlopen("path-to-your-library-file")

print(C.add(3,8))
```

```
from cffi import FFI

ffi = FFI()
ffi.cdef("""
        int add(int, int);
""")

C = ffi.dlopen("rust_functions/target/release/librust_functions.so");

print(C.add(12,8))
```

File Edit View Search Terminal Help

```
[user@lecture-38 addition]$ python run.py
20
[user@lecture-38 addition]$
```

COMPARE AGAINST NUMPY

- Sorting
- Simple map operations

File Edit View Search Terminal Help

```
[user@lecture-38 addition]$ cd ../
[user@lecture-38 ds210]$ mkdir sorting
[user@lecture-38 ds210]$ cd sorting/
[user@lecture-38 sorting]$ cargo new --lib rust_functions
     Created library `rust_functions` package
[user@lecture-38 sorting]$ cd rust_functions/
[user@lecture-38 rust_functions]$
```

File Edit View Search Terminal Help

```
[package]
name = "rust_functions"
version = "0.1.0"
edition = "2021"

# See more keys and their definitions at https://doc.rust-lang.org/cargo/reference/manifest.html

[dependencies]
rayon = "1.5.2"

[lib]
name = "rust_functions"
crate-type = ["dylib"]
```

"Cargo.toml" 13 lines --100%-- 13,22 All

```
use rayon::prelude::*;

#[no_mangle]
pub extern "C" fn sort(pointer:usize, length:usize) -> () {
    let p = unsafe {
        std::mem::transmute::<usize,*mut i32>(pointer)
    };
    let slice = unsafe {
        std::slice::from_raw_parts_mut(p, length)
    };
    slice.sort_unstable();
}

#[no_mangle]
pub extern "C" fn sort_par(pointer:usize, length:usize) -> () {
    let p = unsafe {
        std::mem::transmute::<usize,*mut i32>(pointer)
    };
    let slice = unsafe {
        std::slice::from_raw_parts_mut(p, length)
    };
    slice.par_sort_unstable();
}
~
~
~
~
~
~
~
~
~
~
~
```

File Edit View Search Terminal Help

```
[user@lecture-38 rust_functions]$ cargo build --release
    Updating crates.io index
   Compiling autocfg v1.1.0
   Compiling crossbeam-utils v0.8.8
   Compiling cfg-if v1.0.0
   Compiling lazy_static v1.4.0
   Compiling libc v0.2.125
   Compiling rayon-core v1.9.2
   Compiling scopeguard v1.1.0
   Compiling either v1.6.1
   Compiling memoffset v0.6.5
   Compiling crossbeam-epoch v0.9.8
   Compiling rayon v1.5.2
   Compiling crossbeam-channel v0.5.4
   Compiling num_cpus v1.13.1
   Compiling crossbeam-deque v0.8.1
   Compiling rust_functions v0.1.0 (/home/user/ds210/sorting/rust_functions)
    Finished release [optimized] target(s) in 6.39s
[user@lecture-38 rust_functions]$
```

File Edit View Search Terminal Help

```
[user@lecture-38 rust_functions]$ cd ..
[user@lecture-38 sorting]$
```

```python
import numpy as np
from cffi import FFI
import timeit
import random

def time_it_start():
    return timeit.default_timer()

def time_it_stop(before):
    duration = timeit.default_timer() - before
    print(f"Time: {duration:.3f} s")

def random_array(size=10):
    a = np.arange(size, dtype=np.int32)
    for i in range(size):
        a[i] = random.randint(1,size)
    return a

def python_sort(l):
    print("=== python sort ===")
    t = time_it_start()
    l.sort()
    time_it_stop(t)
    return l

def numpy_sort(arr):
    print("=== numpy sort ===")
    t = time_it_start()
    sorted = np.sort(arr)
    time_it_stop(t)
    return sorted

def rust_sort(arr, C):
    print("=== rust sort ===")
```

```
def rust_sort(arr, C):
    print("=== rust sort ===")
    t = time_it_start()
    C.sort(arr.__array_interface__['data'][0], np.size(arr))
    time_it_stop(t)
    return arr

def rust_sort_par(arr, C):
    print("=== rust sort parallel ===")
    t = time_it_start()
    C.sort_par(arr.__array_interface__['data'][0], np.size(arr))
    time_it_stop(t)
    return arr

ffi = FFI()
ffi.cdef("""
        void sort(uint64_t,uint64_t);
        void sort_par(uint64_t,uint64_t);
""")

C = ffi.dlopen("rust_functions/target/release/librust_functions.so")

array_length = 50 * 1000 * 1000;
arr = random_array(array_length)

out0 = python_sort(arr.tolist())
out1 = numpy_sort(arr.copy())
out2 = rust_sort(arr.copy(), C)
out3 = rust_sort_par(arr.copy(), C)

print(out0[:10],out0[-10:])
print(out1)
print(out2)
print(out3)
```

File Edit View Search Terminal Help

```
[user@lecture-38 sorting]$ python run.py
=== python sort ===
Time: 19.377 s
=== numpy sort ===
Time: 3.111 s
=== rust sort ===
Time: 1.259 s
=== rust sort parallel ===
Time: 0.470 s
[1, 1, 2, 4, 4, 5, 7, 7, 7, 7] [49999986, 49999989, 49999990, 49999990, 49999991, 49999991, 49999995, 49999998, 49999999, 50000000]
[       1        1        2 ... 49999998 49999999 50000000]
[       1        1        2 ... 49999998 49999999 50000000]
[       1        1        2 ... 49999998 49999999 50000000]
[user@lecture-38 sorting]$
```

POTENTIALLY SIMPLER APPROACH

- see crate pyO3
- no covered here today

PURE PYTHON TOO SLOW?

But you still want to use it, at least for your overall data flow. What can you do?

PURE PYTHON TOO SLOW?

But you still want to use it, at least for your overall data flow. What can you do?

- Call one of the off-the-shelf efficient libraries for Python such as `numpy`
  - They are often implemented in C, C++, or another efficient language

PURE PYTHON TOO SLOW?

But you still want to use it, at least for your overall data flow. What can you do?

- Call one of the off-the-shelf efficient libraries for Python such as `numpy`
  - They are often implemented in C, C++, or another efficient language
- Cython
  - Compiles Python code to C

PURE PYTHON TOO SLOW?

But you still want to use it, at least for your overall data flow. What can you do?

- Call one of the off-the-shelf efficient libraries for Python such as `numpy`
  - They are often implemented in C, C++, or another efficient language
- Cython
  - Compiles Python code to C
- Write code yourself in Rust, C, or C++ and create bidings to Python
  - This is what we did today