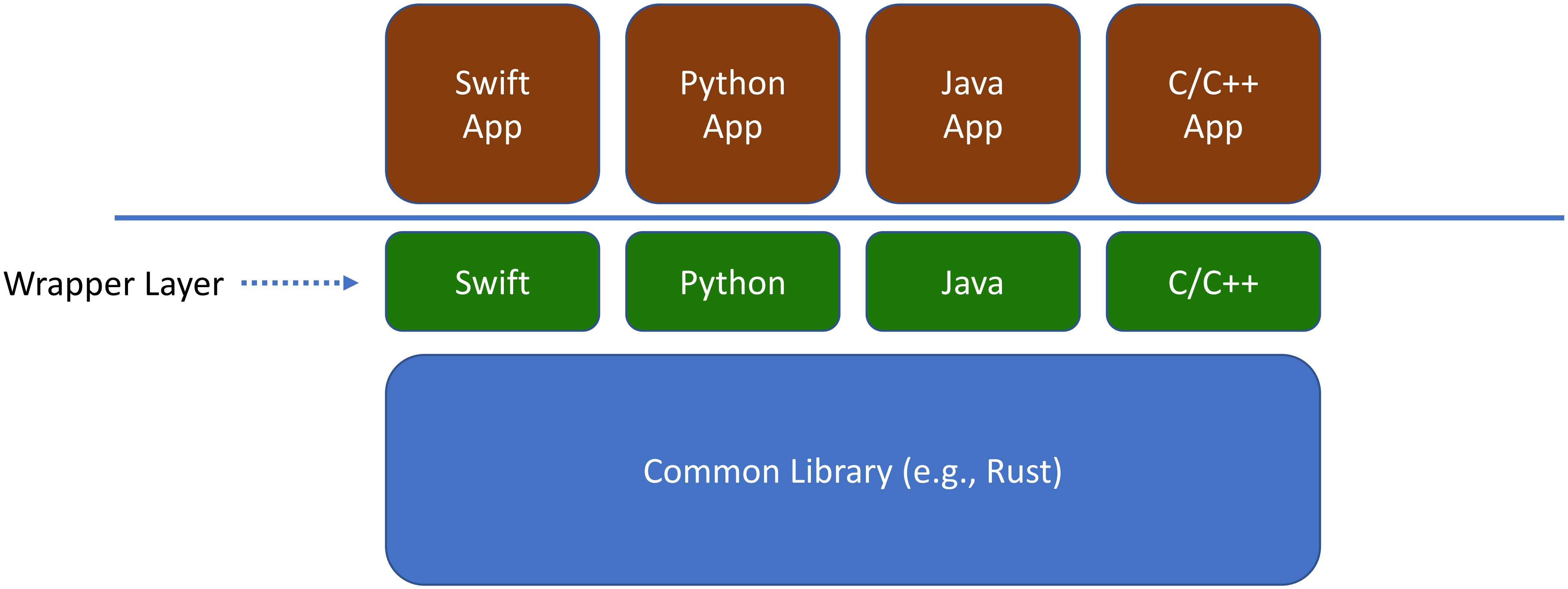
# Methods for Wrapping Rust Code

Steve McCown

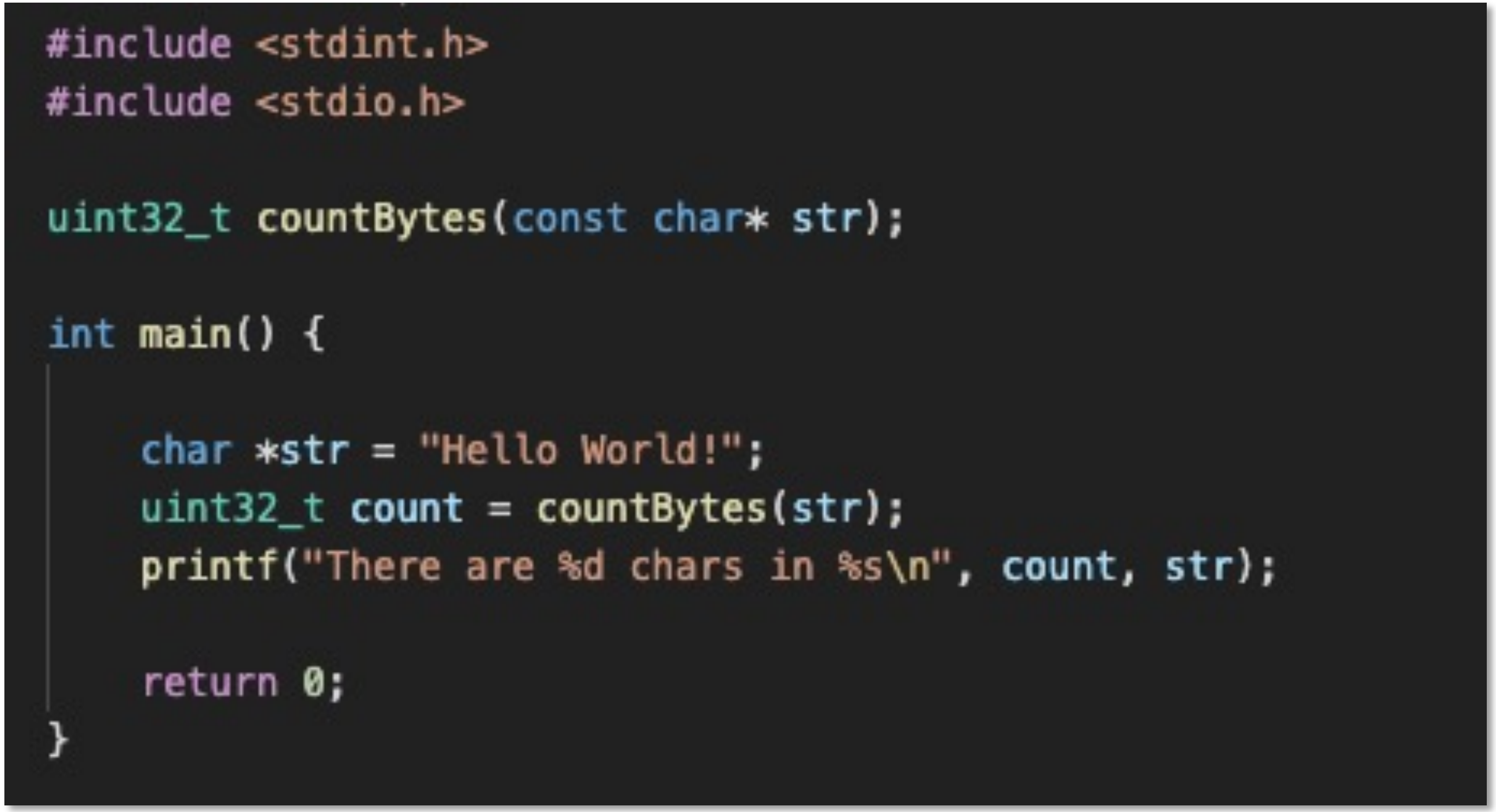
smccown@anonyome.com

25 March 2021

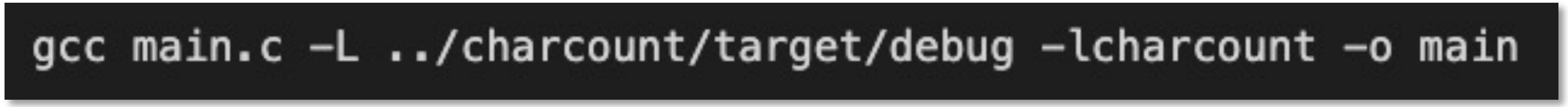
# Common Libraries with Language Wrappers



# Simple C-to-Rust Library Call

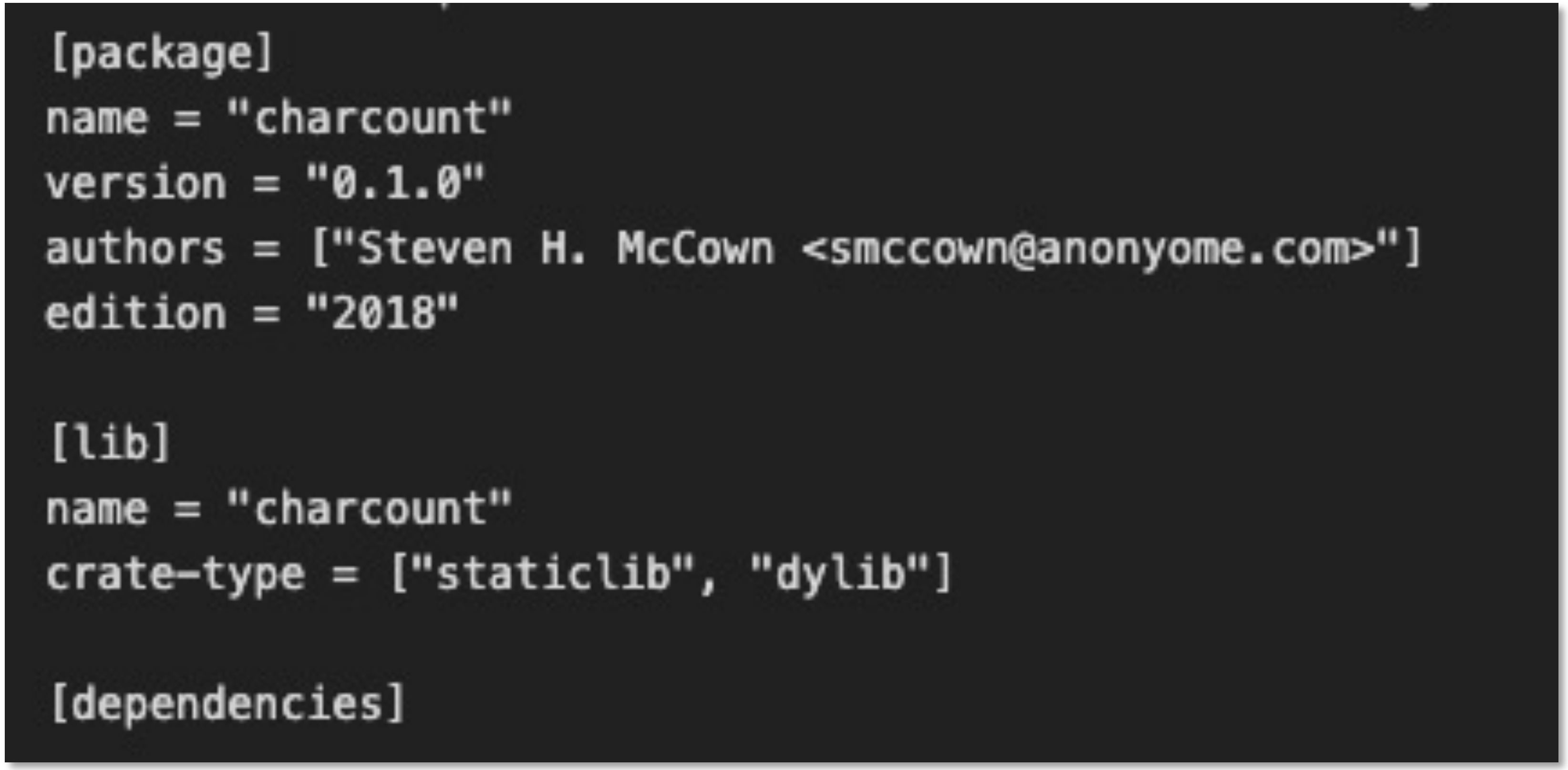


```c
#include <stdint.h>
#include <stdio.h>

uint32_t countBytes(const char* str);

int main() {

    char *str = "Hello World!";
    uint32_t count = countBytes(str);
    printf("There are %d chars in %s\n", count, str);

    return 0;
}
```



```
gcc main.c -L ../charcount/target/debug -lcharcount -o main
```

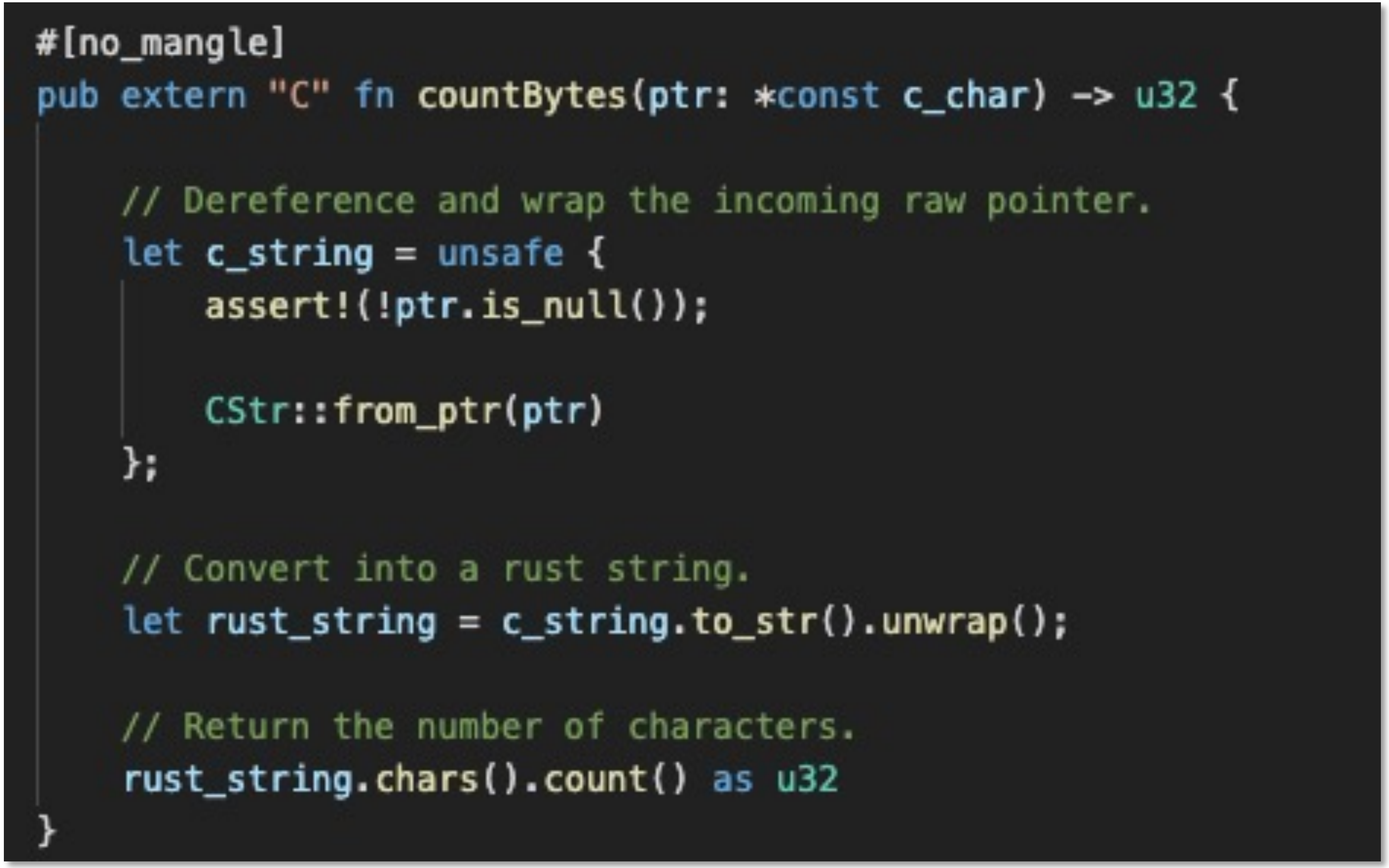
# Create Cargo.toml for Building a Library



```toml
[package]
name = "charcount"
version = "0.1.0"
authors = ["Steven H. McCown <smccown@anonyome.com>"]
edition = "2018"

[lib]
name = "charcount"
crate-type = ["staticlib", "dylib"]

[dependencies]
```

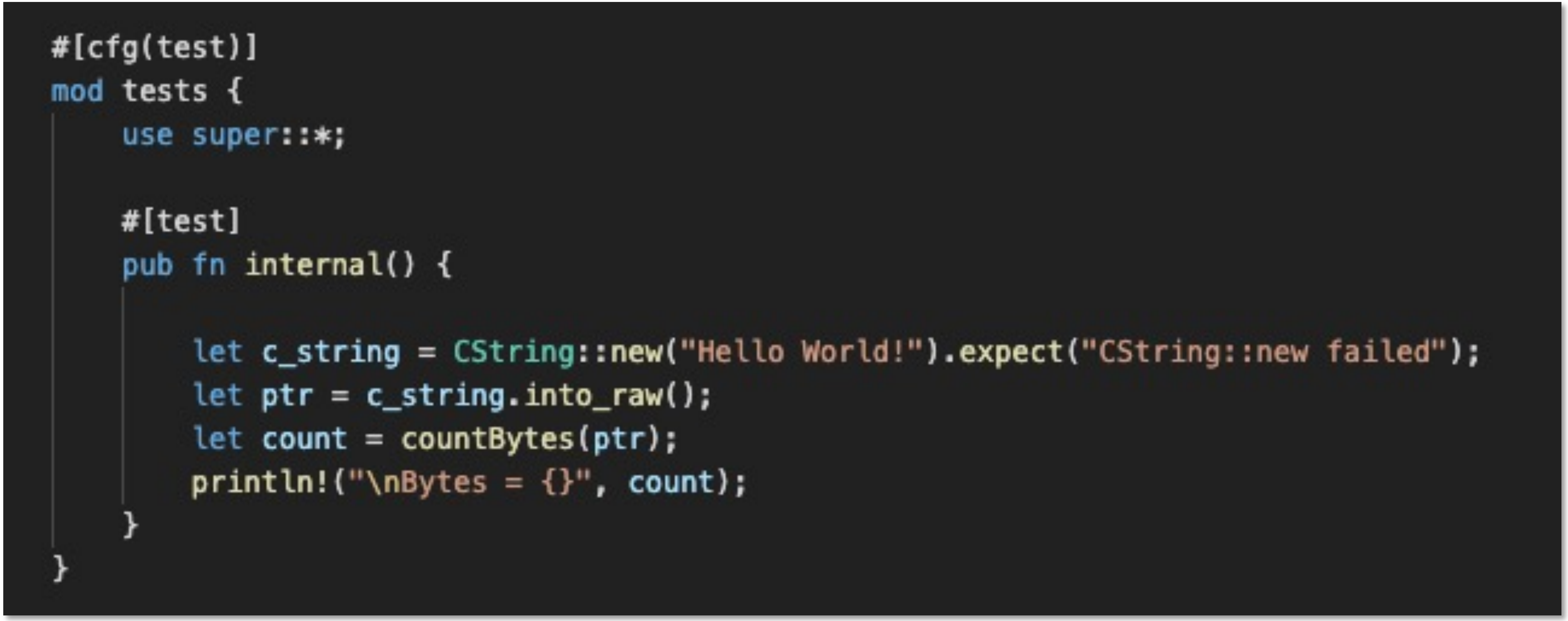
Wrapper Lib

# Create a Rust Library / Wrapper



```rust
#[no_mangle]
pub extern "C" fn countBytes(ptr: *const c_char) -> u32 {

    // Dereference and wrap the incoming raw pointer.
    let c_string = unsafe {
        assert!(!ptr.is_null());

        CStr::from_ptr(ptr)
    };

    // Convert into a rust string.
    let rust_string = c_string.to_str().unwrap();

    // Return the number of characters.
    rust_string.chars().count() as u32
}
```

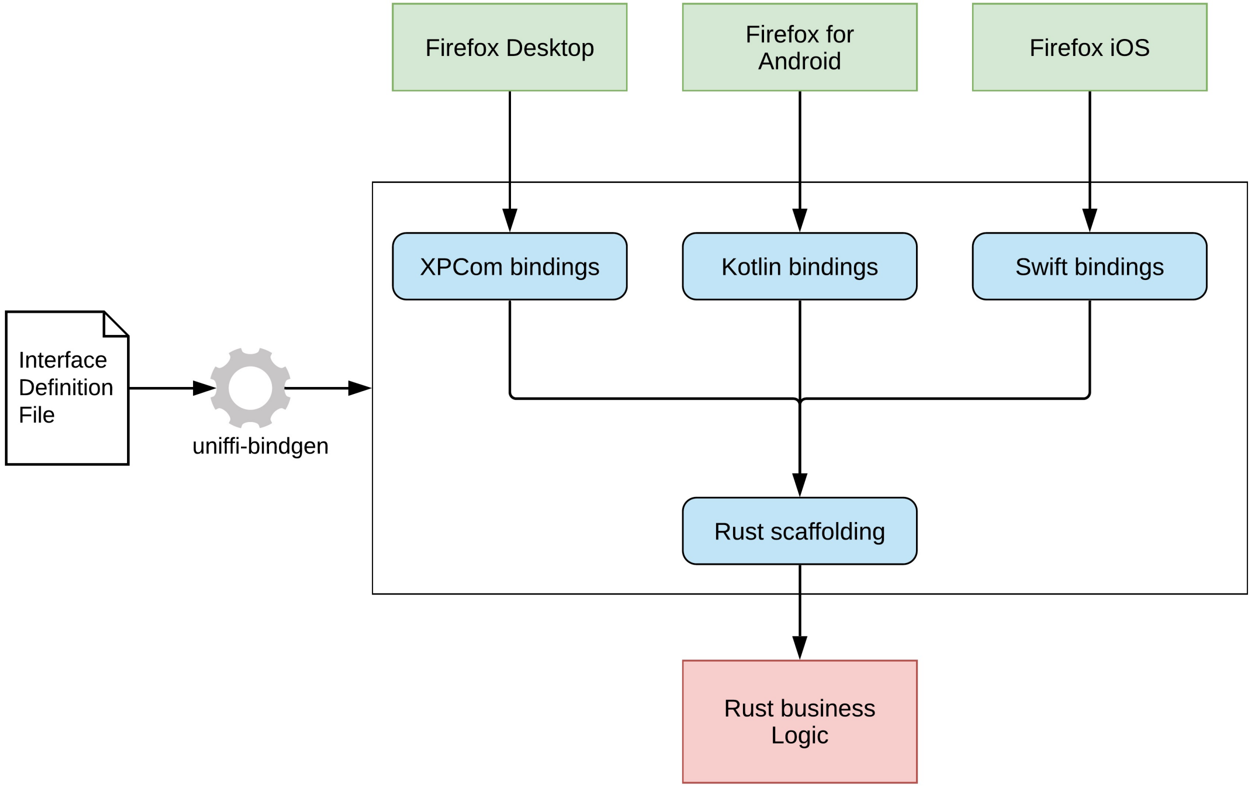
# Testing:  Also More Complex



```
#[cfg(test)]
mod tests {
    use super::*;

    #[test]
    pub fn internal() {

        let c_string = CString::new("Hello World!").expect("CString::new failed");
        let ptr = c_string.into_raw();
        let count = countBytes(ptr);
        println!("\nBytes = {}", count);
    }
}
```

# Uniffi by Mozilla

- Automatically generates foreign-language bindings for Rust libs
- Consolidates business logic into a portable library
- Builds wrappers for
  - Kotlin
  - Swift
  - Python
  - C++
- https://github.com/mozilla/uniffi-rs



# Uniffi:  How it works

1. Create custom Rust library
   - Expose API functions (top-level)
   - Build crate as dylib (in Cargo.toml)

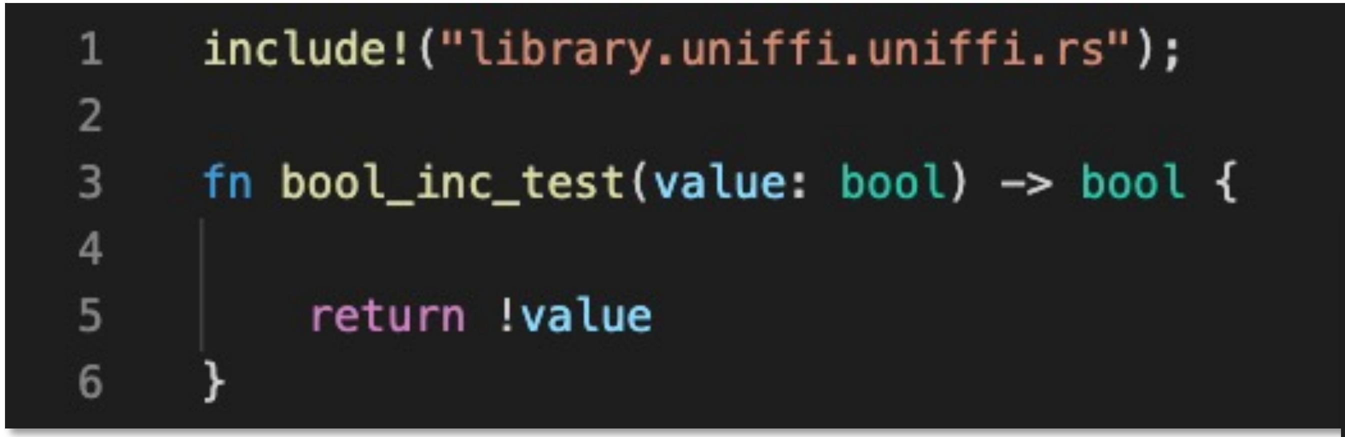
2. Create a UDL representation of API functions
   - Similar to Interface Definition Language (IDL)

3. Create a “Scaffolding” layer
   - Helper code to make foreign-language bindings

4. Create language-specific implementation layer
   - Makes FFI calls look & feel like native code (e.g., Swift FFI feels like native Swift)

5. Import generated code & library into native application

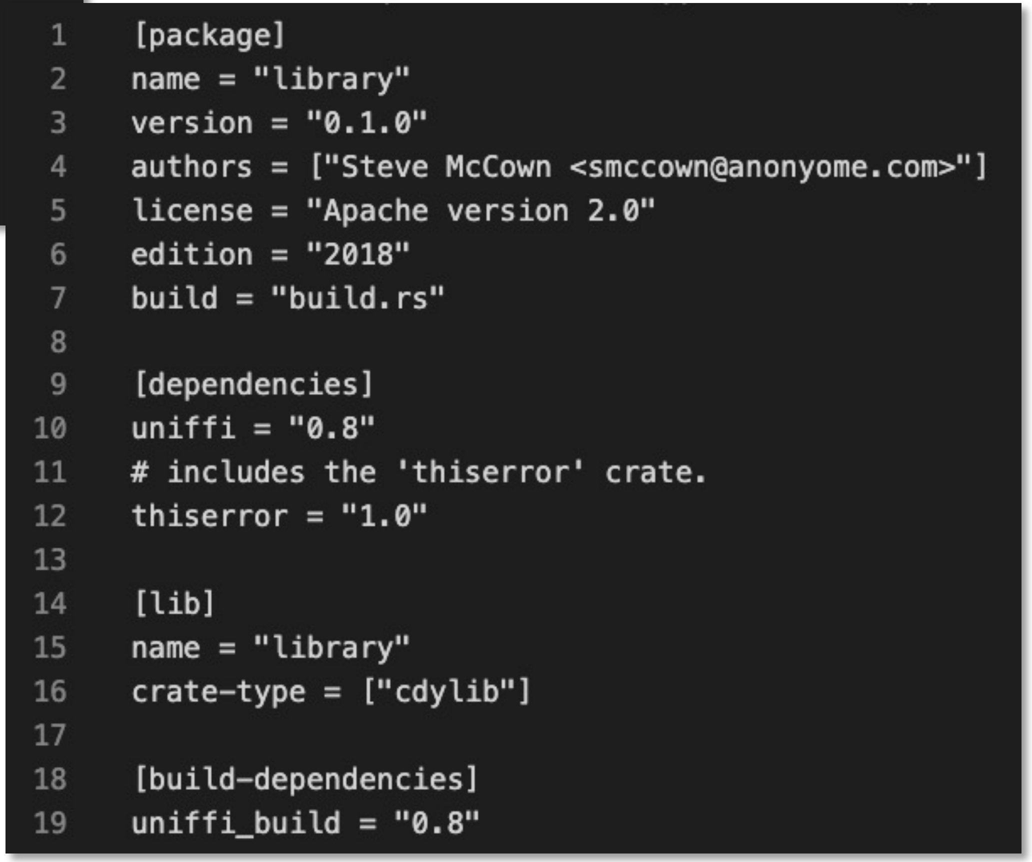
# Create Rust Library



```rust
include!("library.uniffi.uniffi.rs");

fn bool_inc_test(value: bool) -> bool {

    return !value
}
```

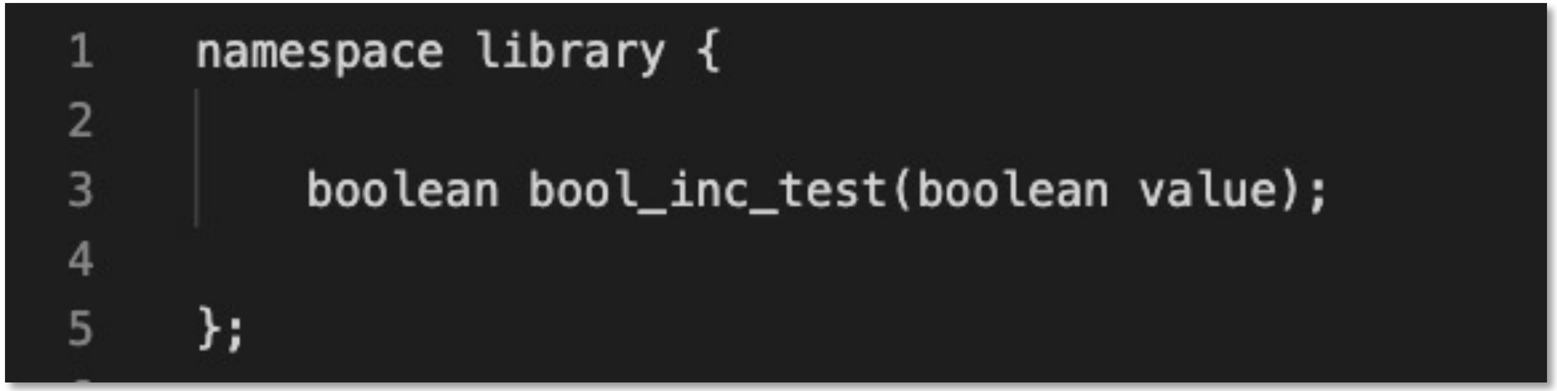
Wrapper Lib

Cargo.toml



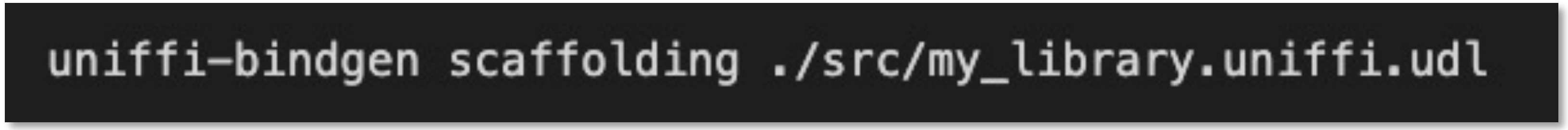
```toml
[package]
name = "library"
version = "0.1.0"
authors = ["Steve McCown <smccown@anonyome.com>"]
license = "Apache version 2.0"
edition = "2018"
build = "build.rs"

[dependencies]
uniffi = "0.8"
# includes the 'thiserror' crate.
thiserror = "1.0"

[lib]
name = "library"
crate-type = ["cdylib"]

[build-dependencies]
uniffi_build = "0.8"
```

# Create UDL for API Functions



```
namespace library {

    boolean bool_inc_test(boolean value);

};
```

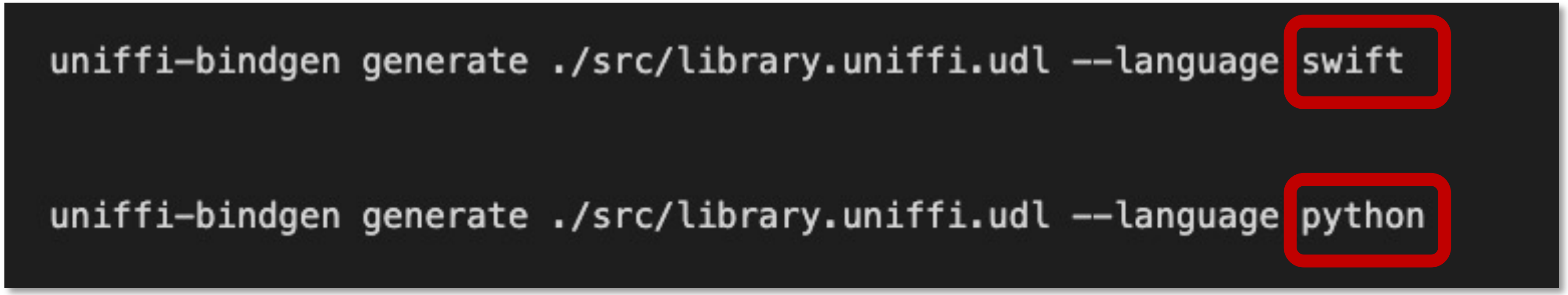
# Generate Scaffolding Layer



```
uniffi-bindgen scaffolding ./src/my_library.uniffi.udl
```

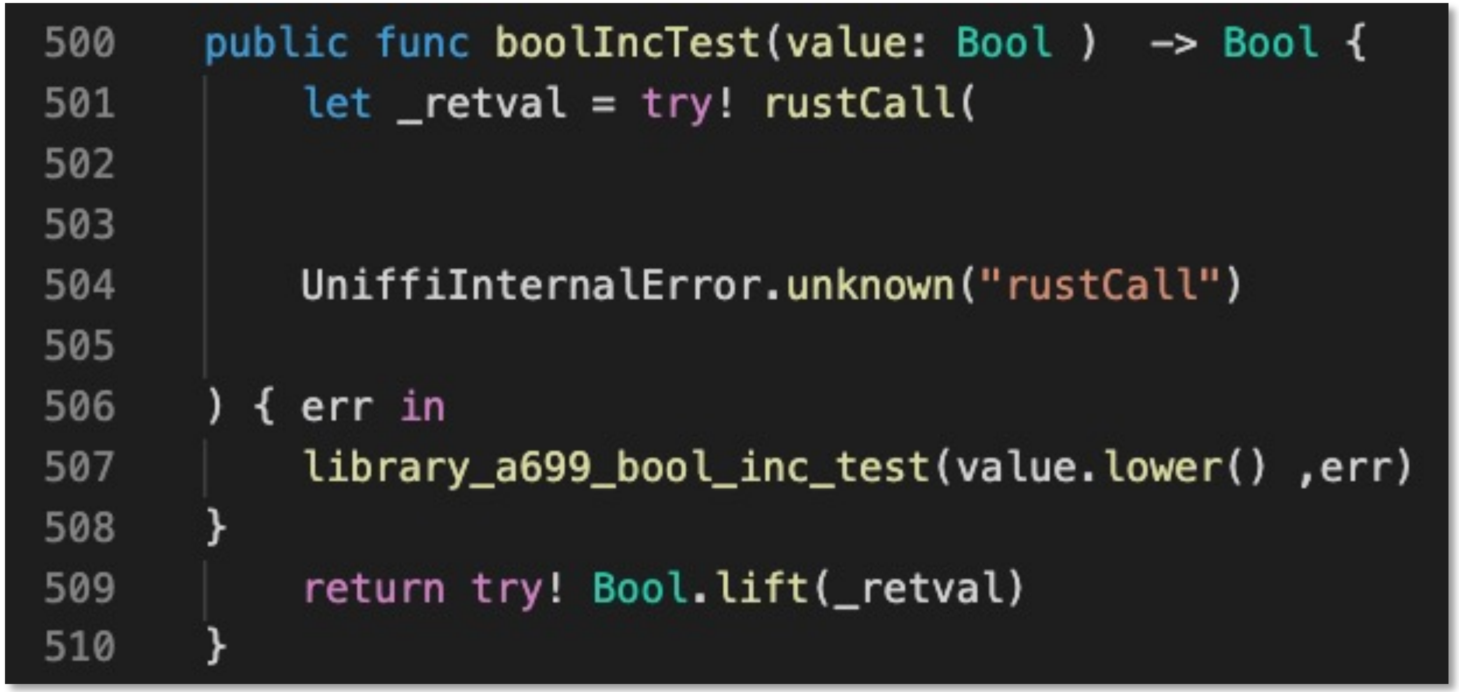
Note:  for details, see the code generated in:   library.uniffi.uniffi.rs

# Generate Language-Specific Interface



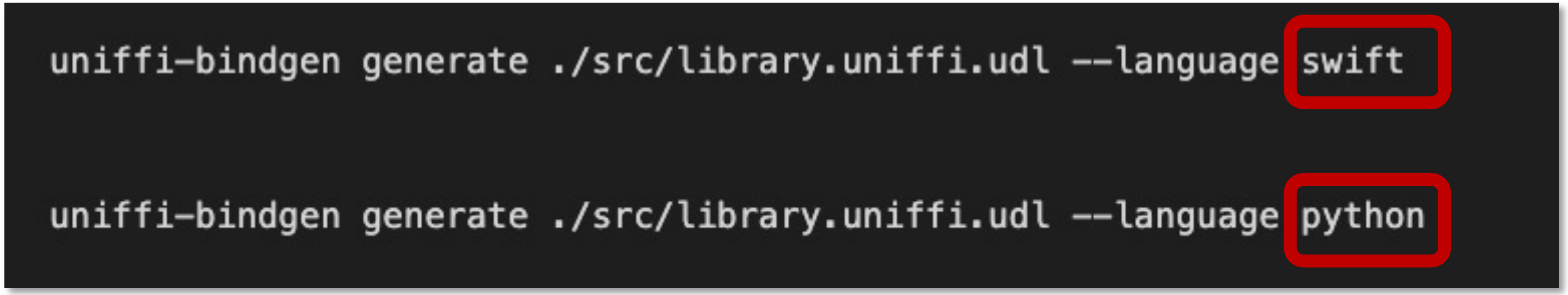
```
uniffi-bindgen generate ./src/library.uniffi.udl --language swift

uniffi-bindgen generate ./src/library.uniffi.udl --language python
```

library.swift



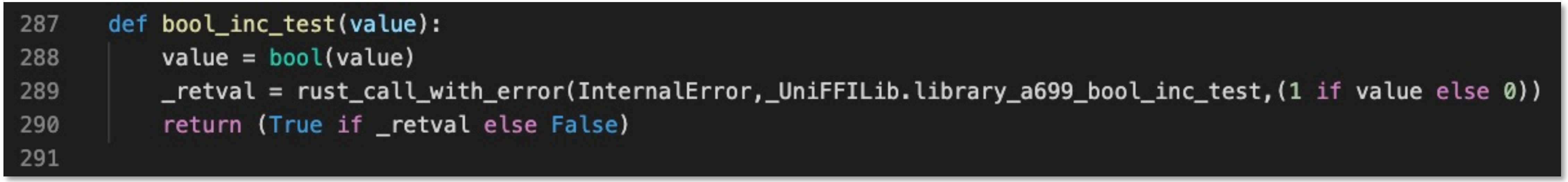
```
public func boolIncTest(value: Bool )  -> Bool {
    let _retval = try! rustCall(


    UniffiInternalError.unknown("rustCall")

) { err in
    library_a699_bool_inc_test(value.lower() ,err)
}
    return try! Bool.lift(_retval)
}
```

# Generate Language-Specific Interface



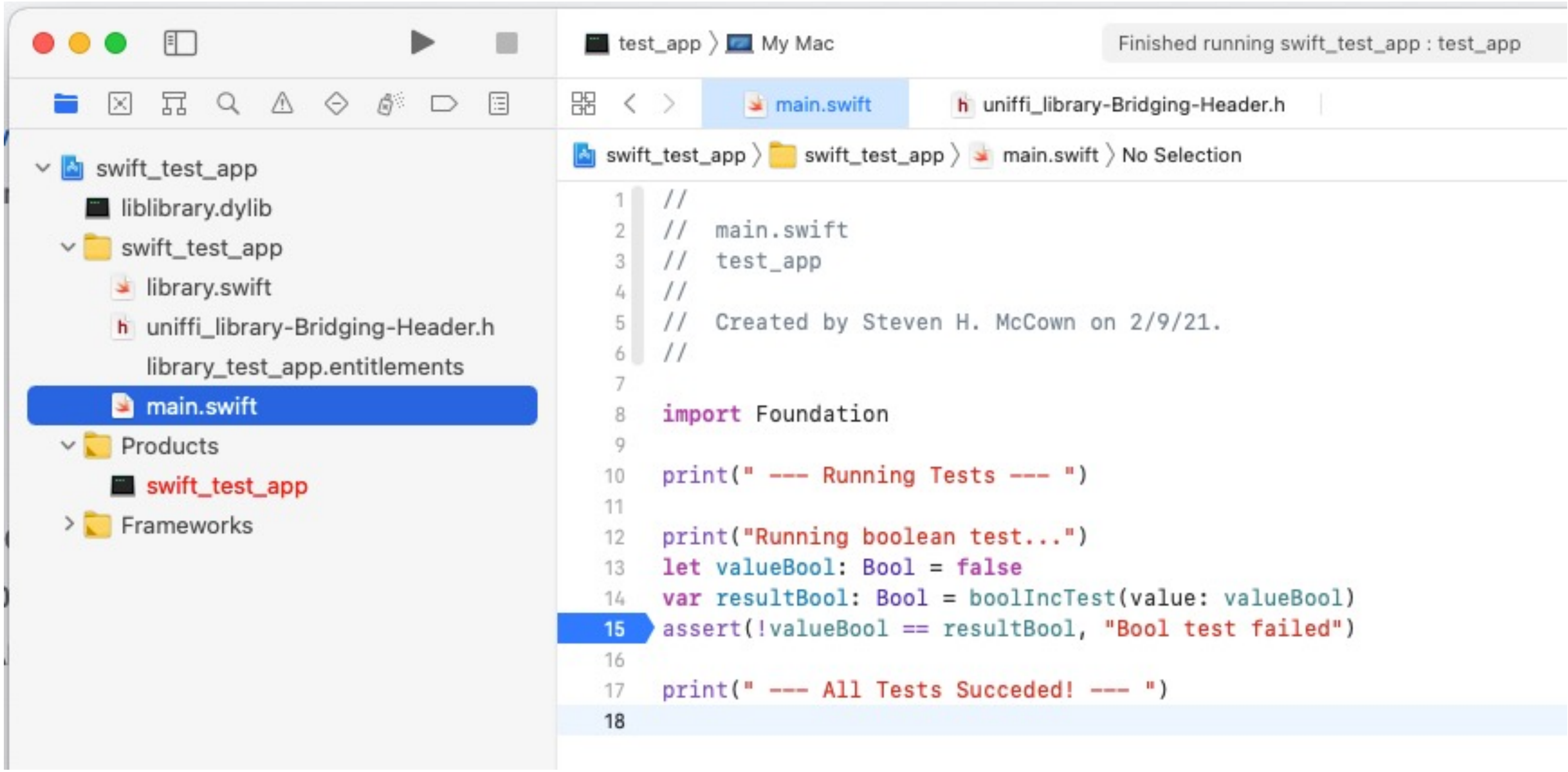
```
uniffi-bindgen generate ./src/library.uniffi.udl --language swift

uniffi-bindgen generate ./src/library.uniffi.udl --language python
```

library.py



```
def bool_inc_test(value):
    value = bool(value)
    _retval = rust_call_with_error(InternalError,_UniFFILib.library_a699_bool_inc_test,(1 if value else 0))
    return (True if _retval else False)

```

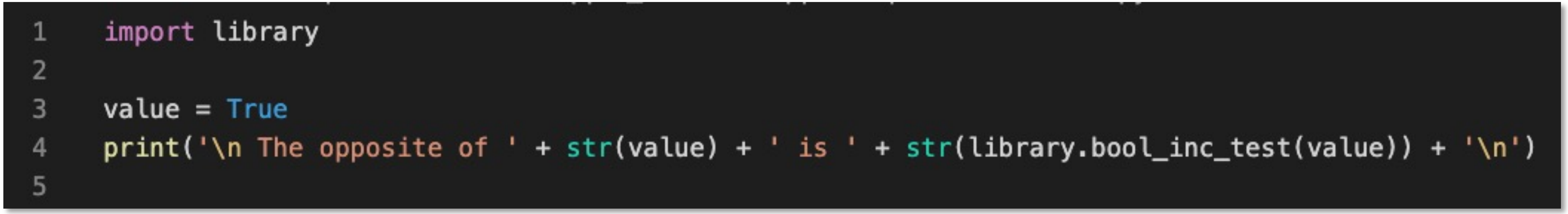
# MacOS Swift App



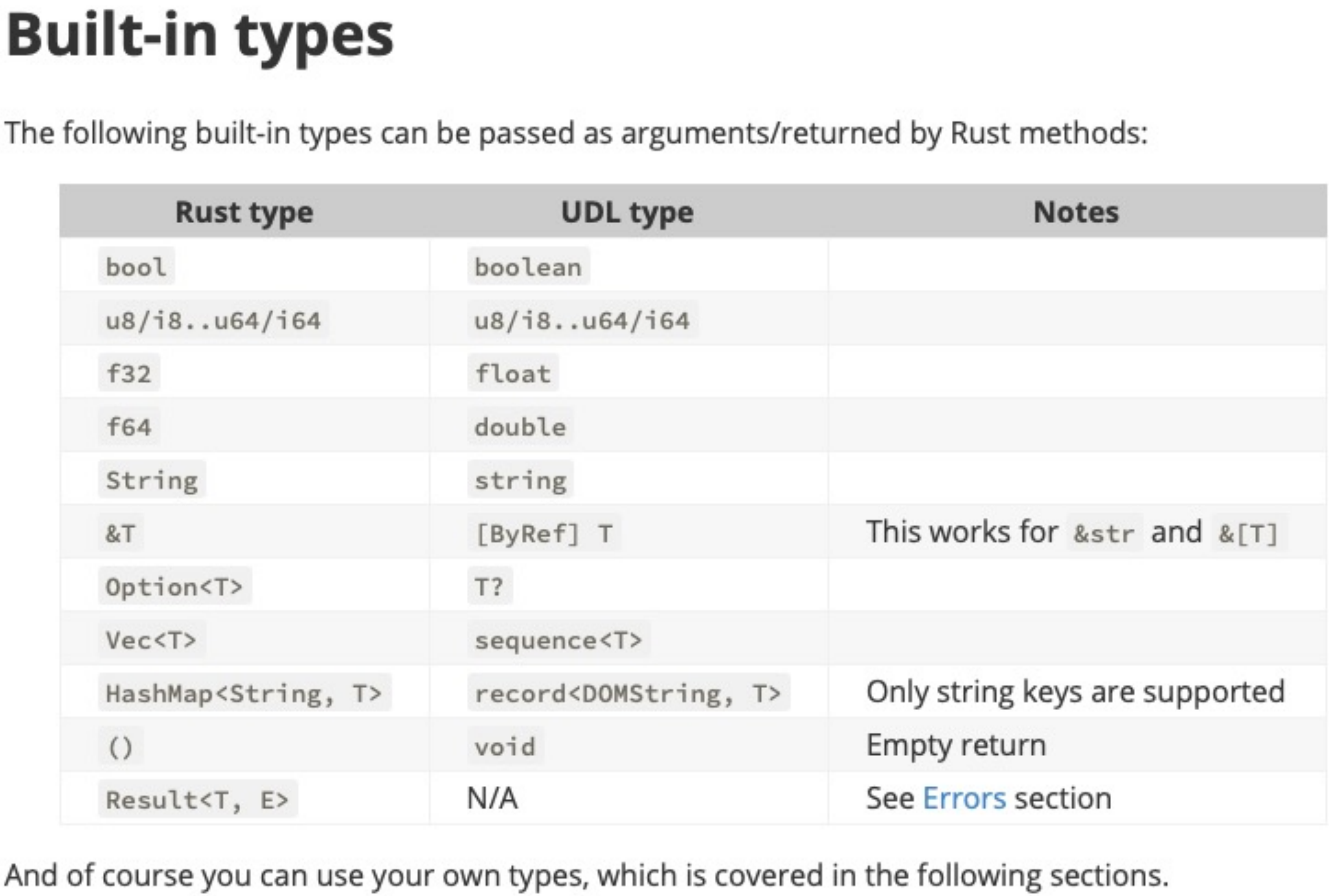
Finished running swift_test_app : test_app

```
//
//  main.swift
//  test_app
//
//  Created by Steven H. McCown on 2/9/21.
//

import Foundation

print(" --- Running Tests --- ")

print("Running boolean test...")
let valueBool: Bool = false
var resultBool: Bool = boolIncTest(value: valueBool)
assert(!valueBool == resultBool, "Bool test failed")

print(" --- All Tests Succeded! --- ")

```

# Python Test App



```python
import library

value = True
print('\n The opposite of ' + str(value) + ' is ' + str(library.bool_inc_test(value)) + '\n')

```



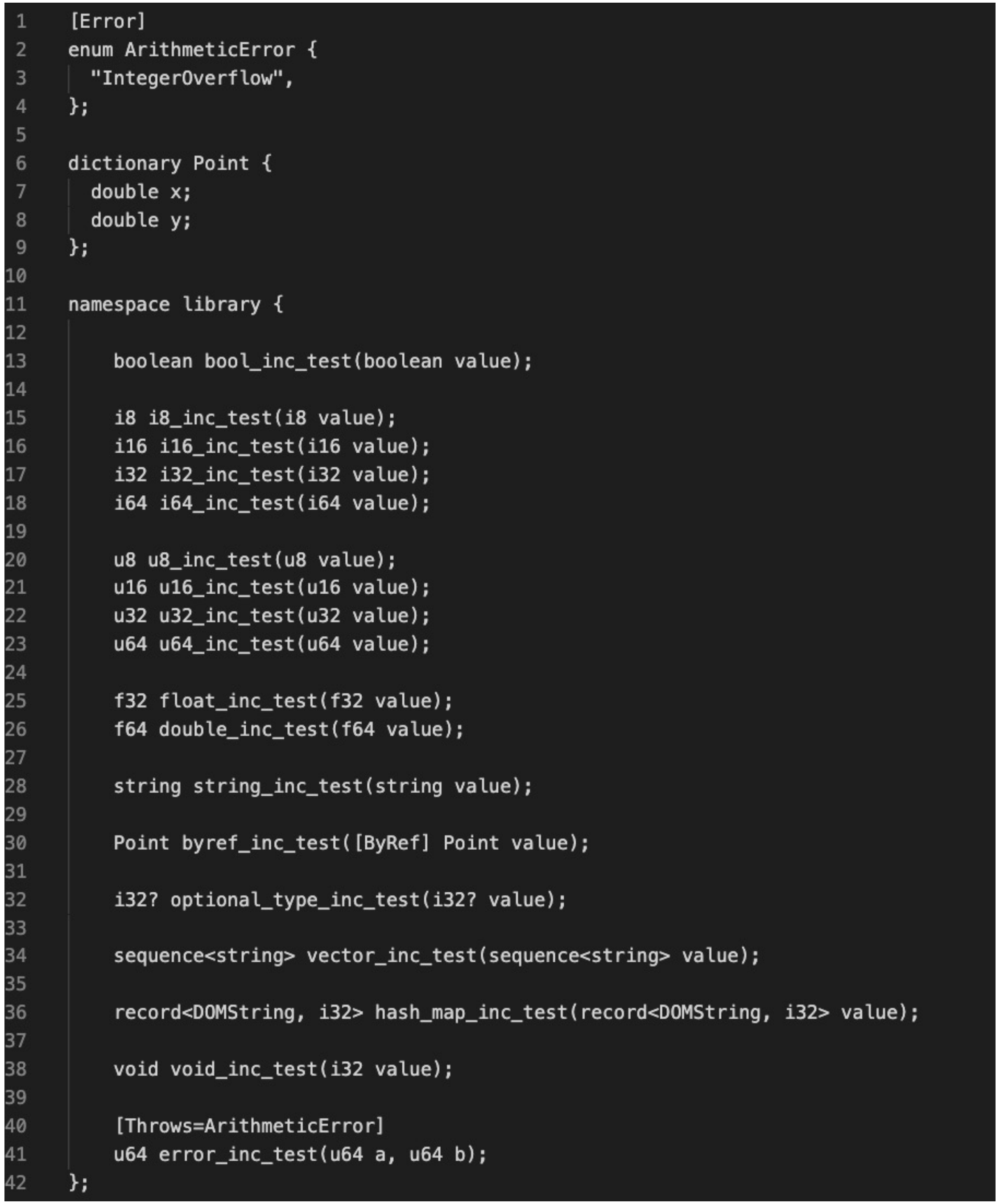
# Built-in types

The following built-in types can be passed as arguments/returned by Rust methods:

| Rust type | UDL type | Notes |
|---|---|---|
| `bool` | `boolean` | |
| `u8/i8..u64/i64` | `u8/i8..u64/i64` | |
| `f32` | `float` | |
| `f64` | `double` | |
| `String` | `string` | |
| `&T` | `[ByRef] T` | This works for `&str` and `&[T]` |
| `Option<T>` | `T?` | |
| `Vec<T>` | `sequence<T>` | |
| `HashMap<String, T>` | `record<DOMString, T>` | Only string keys are supported |
| `()` | `void` | Empty return |
| `Result<T, E>` | N/A | See Errors section |

And of course you can use your own types, which is covered in the following sections.

# Expanded UDL File



```
[Error]
enum ArithmeticError {
  "IntegerOverflow",
};

dictionary Point {
  double x;
  double y;
};

namespace library {

    boolean bool_inc_test(boolean value);

    i8 i8_inc_test(i8 value);
    i16 i16_inc_test(i16 value);
    i32 i32_inc_test(i32 value);
    i64 i64_inc_test(i64 value);

    u8 u8_inc_test(u8 value);
    u16 u16_inc_test(u16 value);
    u32 u32_inc_test(u32 value);
    u64 u64_inc_test(u64 value);

    f32 float_inc_test(f32 value);
    f64 double_inc_test(f64 value);

    string string_inc_test(string value);

    Point byref_inc_test([ByRef] Point value);

    i32? optional_type_inc_test(i32? value);

    sequence<string> vector_inc_test(sequence<string> value);

    record<DOMString, i32> hash_map_inc_test(record<DOMString, i32> value);

    void void_inc_test(i32 value);

    [Throws=ArithmeticError]
    u64 error_inc_test(u64 a, u64 b);
};
```

# Questions?