

The slide features a light orange background with a dark teal border. The border is decorated with various line-art icons in teal, orange, and pink. These icons include laboratory glassware like flasks, beakers, and a graduated cylinder; scientific symbols like a radiation warning sign and a hexagon; and general science symbols like a magnifying glass, a ruler, and molecular structures. Plus signs and starburst symbols are also scattered throughout the border.

# ***Dart FFI and Communicating with Rust***

(Crash course, safety not guaranteed)



# Agenda

## 01

### Dart FFI

What is FFI?

You know the Dart.  
So do I.



## 02

### Rust

What is Rust?

For real.

## 03

### Demo

How to combine these  
two together?







01

# Dart FFI

What is it and why would  
someone do this to  
themselves?



# What is FFI?

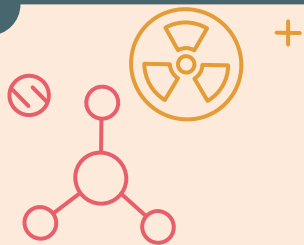


FFI Stands for “Foreign Function Interface”.

They refer to:

- Tools
- APIs
- Mechanisms

providing a way to call functions in native C libraries and allow seamless communication and interoperability.



# Why FFI?



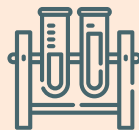
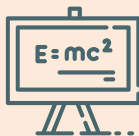
**Access native  
libraries**



**Write  
platform-specific  
c code**



**Improve  
performance**



**Cross-platform  
development**

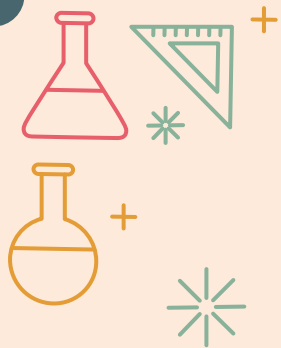


**Bindings for  
existing  
libraries**



**Experimental  
features**





# Dart ffi





# How to use Dart ffi?



**Not Yet**





02

# Rust

It's so beautiful  
I am going to cry







# What and Why of Rust



**History**



**Memory Safety**



**Concurrency**



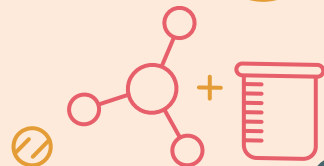
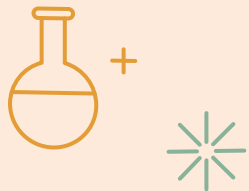
**Syntax**

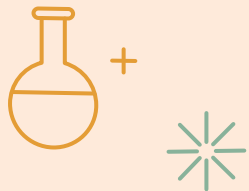


**Extensive Tooling and Ecosystem**



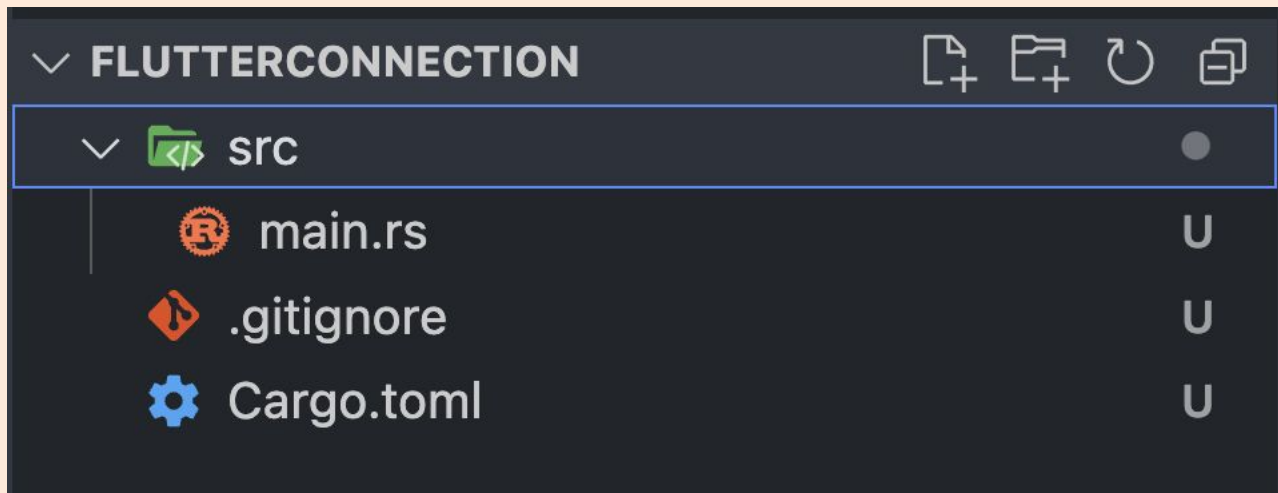
# Simple Application on Rust

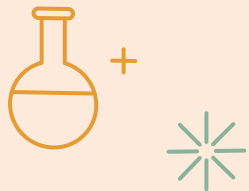




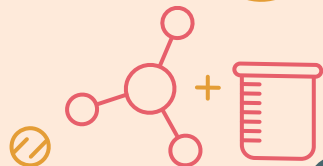
```
cargo new flutterconnection
```

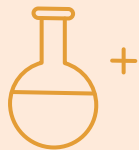






```
fn main() {  
    println!("Hello, world!");  
}
```



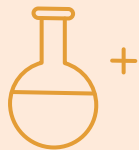


+



```
fn main() {  
    let name: &str = "Salih";  
    let age: u32 = 30;  
  
    println!("My name is {} and I am {} years old.", name, age);  
}
```

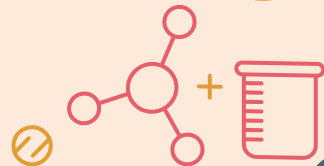




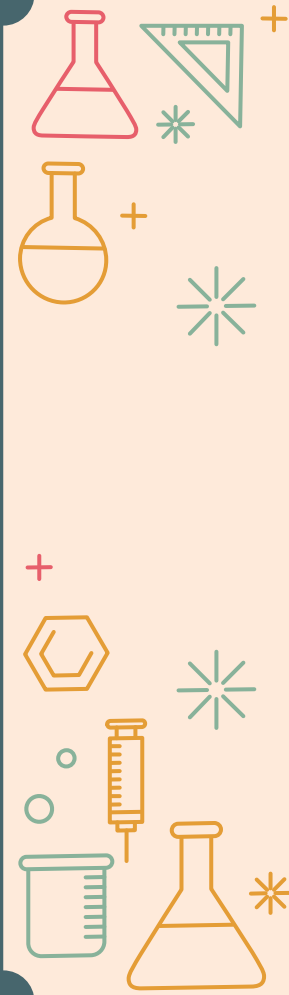
+



\*



```
fn main() {  
    let name = "Salih";  
    let age: u32 = 30;  
  
    println!("My name is {} and I am {} years old.", name, age);  
}
```

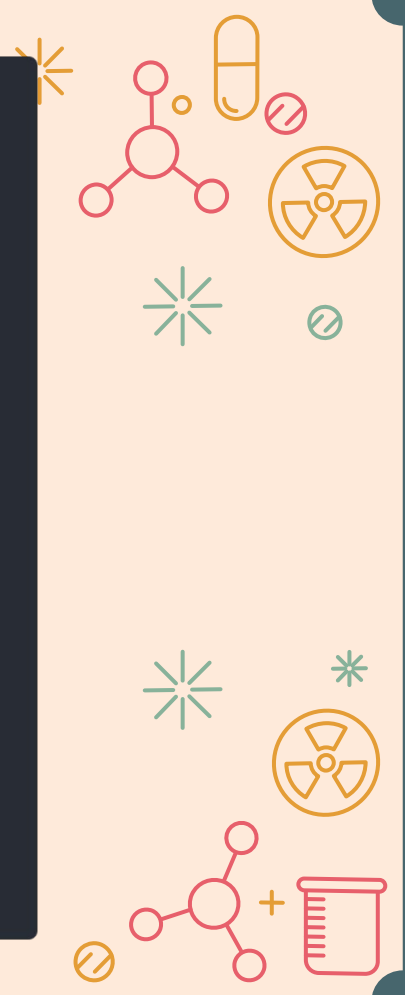


```
fn main() {
    let name = "Salih";
    let age = 30;
    let is_crazy = true;

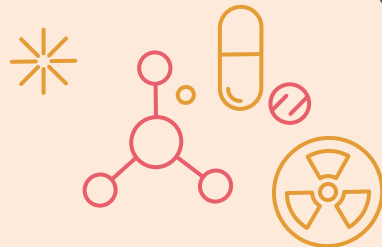
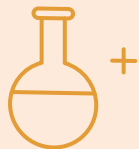
    match name {
        "Salih" => println!("Hello, Salih!"),
        "Gulcin" => println!("Hello, Gulcin!"),
        _ => println!("Hello, stranger!"),
    }

    match age {
        0..=17 => println!("You can only drink in Germany."),
        18..=30 => println!("You can drink alcohol"),
        _ => println!("You can drink alcohol with a default format."),
    }

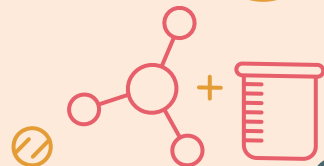
    if is_new {
        println!("Salih is crazy!");
    } else {
        println!("Salih is still crazy but pretends to be normal!");
    }
}
```





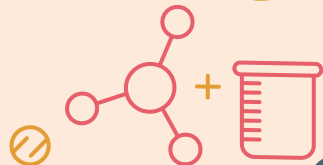


```
fn print_introduction() {  
    println!("My name is Salih and I am 30 years old.");  
}
```





```
fn main() {  
    print_introduction();  
}
```





```
msalihg@bcd074760e20 flutterconnection % cargo run
  Compiling flutterconnection v0.1.0
  (/Users/msalihg/Desktop/development/projects/flutterconnection)
    Finished dev [unoptimized + debuginfo] target(s) in 0.32s
    Running `target/debug/flutterconnection`
My name is Salih and I am 30 years old.
```

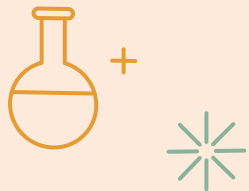


03

# Demo

In the best way possible





```
dart create fc_dart
```





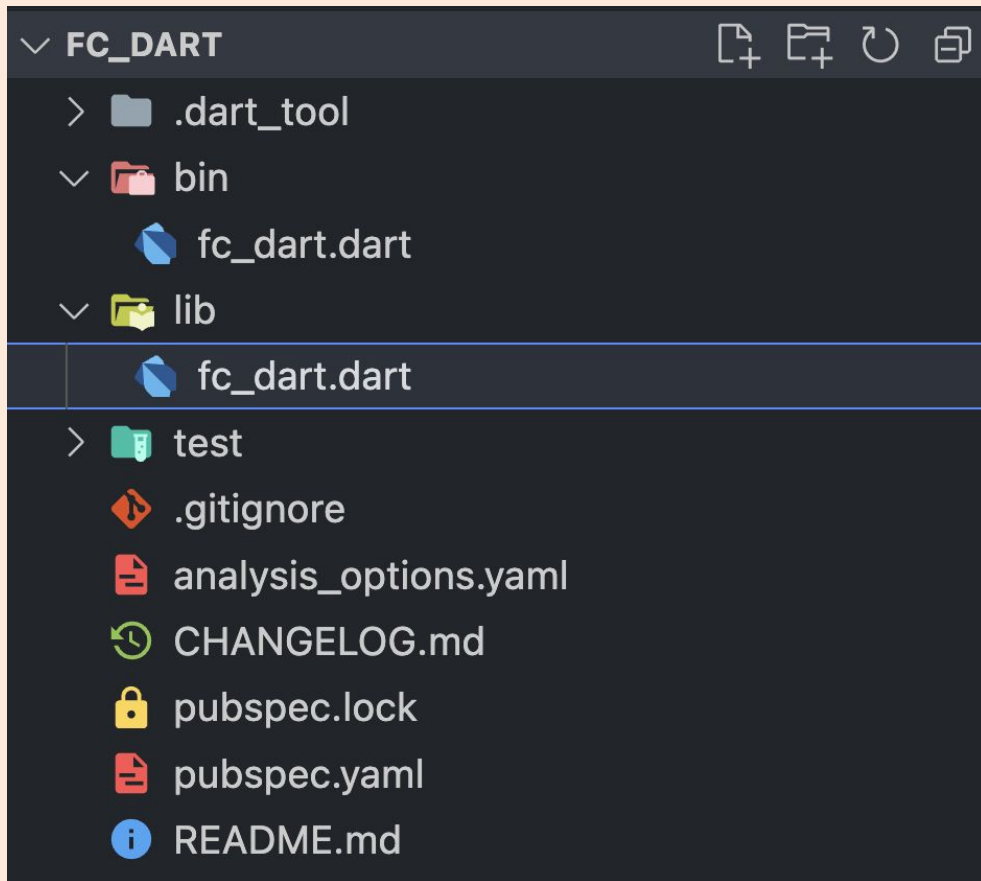
```
msalihg@bcd074760e20 projects % dart create fc_dart
Creating fc_dart using template console...
```

```
.gitignore
analysis_options.yaml
CHANGELOG.md
pubspec.yaml
README.md
bin/fc_dart.dart
lib/fc_dart.dart
test/fc_dart_test.dart
```

```
Running pub get... 3.5s
Resolving dependencies...
Changed 46 dependencies!
```

```
Created project fc_dart in fc_dart! In order to get started, run the following commands:
```

```
cd fc_dart
dart run
```





```
import 'dart:ffi' as ffi;

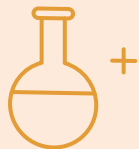
// ignore: camel_case_types
typedef print_introduction = ffi.Void Function();

typedef Introduction = void Function();

void introduce() {
  final libraryPath =
    'flutterconnection/target/debug/libprint_introduction.dylib';
  final dylib = ffi.DynamicLibrary.open(libraryPath);
  final Introduction introduction = dylib
    .lookup<ffi.NativeFunction<print_introduction>>('print_introduction')
    .asFunction();

  introduction();
}
```

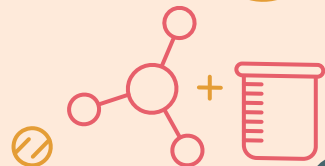


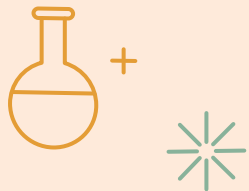


```
#[no_mangle]
pub extern "C" fn print_introduction() {
    println!("My name is Salih and I am 30 years old.");
}
```



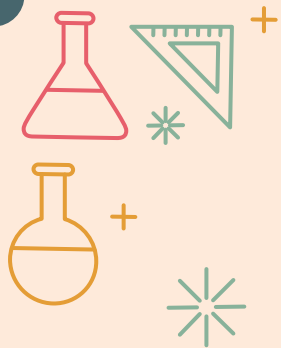
```
[lib]  
name="print_introduction"  
crate-type = ["cdylib"]
```





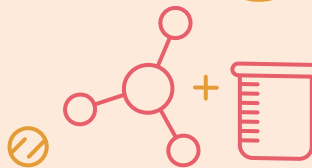
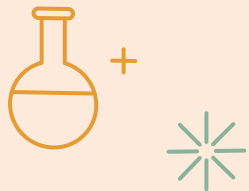
```
• • •  
cargo build
```





My name is Salih and I am 30 years old.  
Exited





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



cargo build








5  
Exited





# How About Complex Scenarios?



The background is a light orange color with various chemistry-related icons scattered around. In the top left, there is a conical flask with liquid, a right-angled triangle ruler, and a plus sign. Below that is a round-bottom flask with liquid and a plus sign. To the right of the central code block, there is a starburst, a molecular structure diagram, a pill, a radiation symbol, and a crossed-out circle. In the bottom left, there is a plus sign, a benzene ring, a syringe, a beaker, and a conical flask with liquid and a starburst. In the bottom right, there is a radiation symbol, a molecular structure diagram, a plus sign, and a beaker.

```
use std::ffi::{CStr, CString};
use serde_json;

#[no_mangle]
pub extern "C" fn process_json(json_ptr: *const libc::c_char) {
    unsafe {
        let c_str = CStr::from_ptr(json_ptr);
        let json_string = c_str.to_str().unwrap();
        let deserialized: serde_json::Value = serde_json::from_str(json_string).unwrap();

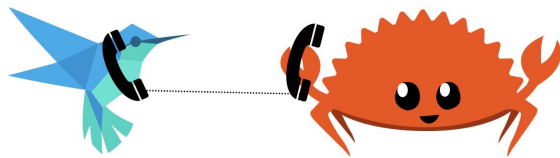
        // Perform some processing on the deserialized object
        // For simplicity, we'll just print it here
        println!("Received JSON: {:?}", deserialized);
    }
}
```



flutter\_rust\_bridge

## flutter\_rust\_bridge: High-level memory-safe binding generator for Flutter/Dart <-> Rust

crates.io v1.77.1 pub v1.77.1 stars 2.6k CI failing Post-Release failing code quality A



Want to combine the best between [Flutter](#), a cross-platform hot-reload rapid-development UI toolkit, and [Rust](#), a language empowering everyone to build reliable and efficient software? Here it comes!

### Advantages

- **Memory-safe:** Never need to think about malloc/free.
- **Feature-rich:** `enum`s with values, platform-optimized `Vec`, possibly recursive `struct`, zero-copy big arrays, opaque types on arbitrary structs/classes, `Stream` (iterator) abstraction, error (`Result`) handling, cancellable tasks, concurrency control, and more. See full features [here](#).
- **Async programming:** Rust code will never block the Flutter. Call Rust naturally from Flutter's main isolate (thread); sync mode also equally supported.
- **Lightweight:** This is not a huge framework that includes everything, so you are free to use your favorite Flutter and Rust libraries. For example, state-management with Flutter library (e.g. MobX) can be elegant and simple (contrary to implementing in Rust); implementing a photo manipulation algorithm in Rust will be fast and safe (country to implementing in Flutter).



# Resources

## Socials



## Docs



# Thanks!

## Questions?



@salihgueler

