Introduction
-
Welcome to Crabland

Why Rust

- Safety
- Speed
- Concurrency
- Expressivity
- No garbage collection and no manual memory management! (lifetime analysis)
- Zero-cost abstractions
- Very complete toolchain and build system (rustfmt, cargo, rustup…)
- Zero setup cross compilation
- Strong ecosystem and tons of libraries at your fingertips!
- Integrated unit testing

```c
#include <stdio.h>

int main() {
    printf("Hello, world!\n");

    return 0;
}
```

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

```c
#include <stdio.h>

int main() {
    const char *elite = "GISTRE";

    printf("Hello, %s!\n", elite);

    return 0;
}
```

```rust
fn main() {
    let elite = "GISTRE";

    println!("Hello, {}!", elite);
}
```

- Easy interoperability with C
- Suited for embedded/system programming
- Suited for high level applications

- Functional *AND* Object Paradigm
    - (But we'll see the Object parts later…)
- Everything is an expression
- Iterators
- Method chaining
- ...

```rust
fn sum_loop(start: i32, end: i32) -> i32 {
    let mut acc = 0;
    for i in start..end {
        acc += i;
    }

    return acc;
}
```

```
fn sum_loop(start: i32, end: i32) -> i32 {
    let mut acc = 0;
    for i in start..end {
        acc += i;
    }

    acc
}
```

```rust
fn sum_sum(start: i32, end: i32) -> i32 {
    (start..end).sum()
}
```

```rust
fn sum_fold_left(start: i32, end: i32) -> i32 {
    (start..end).fold(0, |acc, elt| acc + elt)
}
```

```
fn sum_vec(values: Vec<(i32, i32)>) -> Vec<i32> {
    values.iter()
        .map(|(start, end)| sum_fold_left(*start, *end))
        .collect()
}
```

```rust
fn abs(value: i32) -> i32 {
    let result = if value < 0 {
        -value
    } else {
        value
    };
    result
}
```

- No *NULL* pointer (C)
    - *Option<T>: Some<T> | None* (Rust)
    - *Maybe<T>: Just<T> | Nothing* (Haskell)
- No *NullReferenceException* (Garbage (collected) languages like Java, C#...)
- No exceptions at all, actually
    - But more complete error handling than C/C++
        - *Result<T, E>: Ok<T> | Err<E>*

```c
#include <stdio.h>

/**
 * Return NULL on error, the file otherwise
 */
FILE *open_read(const char *path) {
    FILE *file = fopen(path, "r");

    return file;
}
```

```rust
use std::fs::File;

fn open_read(path: &str) -> Option<File> {
    let file_result = File::open(path);

    match file_result {
        Ok(file) => Some(file),
        Err(_) => None,
    }
}
```

- Immutability by default
- Borrow-checker

```
int main() {
    int b = 1;
    b = 2; // Ok

    const int a = 1;
    a = 2; // Error
}
```

```rust
fn main() {
    let a = 1;
    a = 2; // Error

    let mut b = 1;
    b = 2; // Ok
}
```

```
> cargo build
error[E0384]: cannot assign twice to immutable variable `a`
 --> src/main.rs:3:5
  |
2 |     let a = 1;
  |         -
  |         |
  |         first assignment to `a`
  |         help: make this binding mutable: `mut a`
3 |     a = 2;
  |     ^^^^^ cannot assign twice to immutable variable
```

GISTRE

Some C/C++ footguns

```c
#include <stdio.h>
#define BUF_SIZE 256

int main() {
    char input[BUF_SIZE] = { 0 };
    fgets(input, BUF_SIZE, stdin);
    printf(input);

    return 0;
}
```

```
> ./a.out
Coucou GISTRE!
Coucou GISTRE!
> ./a.out
%d %p %d
1881481316 (nil) 882578089
```

```
use std::io;
const BUF_SIZE: usize = 256;

fn main() {
    let mut input = String::with_capacity(BUF_SIZE);
    io::stdin().read_line(&mut input);

    println!(input);
}
```

```
> cargo build
   Compiling hello_world v0.1.0 (/tmp/hello_world)
error: format argument must be a string literal
  --> src/main.rs:10:14
   |
10 |     println!(input);
   |              ^^^^^
   |
help: you might be missing a string literal to format with
   |
10 |     println!("{}", input);
   |              ^^^^^
   |
```

```rust
use std::io;
const BUF_SIZE: usize = 256;

fn main() {
    let mut input = String::with_capacity(256);
    io::stdin().read_line(&mut input);

    println!(input); // Wrong
}
```

```rust
use std::io;
const BUF_SIZE: usize = 256;

fn main() {
    let mut input = String::with_capacity(256);
    io::stdin().read_line(&mut input);

    println!(input); // Wrong
    println!("{}", input); // OK!
}
```

```
int main(void) {
    int buffer[20];

    printf("%d", buffer[2500]);

    return 0;
}
```

```
> gcc main.c # No error...
> ./a.out
[2]    40595 segmentation fault (core dumped)  ./a.out
```

```rust
fn main() {
    let buffer: [i32; 20] = [0; 20];

    println!("{}", buffer[2500]);
}
```

```
> cargo build
   Compiling hello_world v0.1.0 (/tmp/hello_world)
error: this operation will panic at runtime
 --> src/main.rs:4:20
  |
4 |     println!("{}", buffer[2500]);
  |                    ^^^^^^^^^^^^ index out of bounds:
the length is 20 but the index is 2500
  |
  = note: `#[deny(unconditional_panic)]` on by default
```

```cpp
#include <vector>
#include <iostream>
#include <numeric>

int main(void) {
    std::vector<int> vector = { 2, 3, 4, 5 };
    int s = std::accumulate(vector.begin(), vector.end(), 0);
    std::cout << s << std::endl;
    return 0;
}
```

```rust
fn main() {
    let vector = vec![2, 3, 4, 5];
    let sum: i32 = vector.iter().sum();

    println!("{}", sum);
}
```