Rust closures

A simple example

```
let x = 2.0;
let a = 1.0;
let b = 2.0;

let f = |x| a * x + b;
// If we remove this line then type inference stops working
println!("f({}) = {}", x, f(x));
```

Why is it 🔥 fast?

We don't box closures! They can be inlined as well

I am not gonna sugarcoat it

```
struct FStruct<'a> {
    a: &'a f64,
    b: &'a f64,
}

impl FStruct<'_> {
    fn call(&self, x: f64) -> f64 {
        self.a * x + self.b
    }

let x = 3.0;
println!("f({}) = {}", x, FStruct { a: &2.0, b: &2.0 }.call(x));
```

Calling a closure is just calling a method on a struct

Moving

```
let a = "hello".to_string();
let fs = || {
    let b = a;
    println!("we got {}", b);
};

println!("{}", a) // illegal
```

can be also made explicit with `move`

Moving with copy

```
let f = move |x| a * x + b;
```

Fns

There are three types of them and they represent how the closure interacts with the captured variables.

`Fn`

is a `&self` method

```
let f = |x| a * x + b
```

`FnMut`

is a `&mut self` method

```
let mut fs = || {
    a.push_str("hi");
}
```

FnOnce

is a `self` method

```
let a = "hi".to_string();
let fs = || {
    let b = a;
}
```

Question time

```
let mut fs: ??? = |a: String| {
    a.push_str("dupa");
    println!("{a}");
}
```

Understanding closures

```
let mut a = "hello".to_string();
let mut fs = move || {
    a.push_str(" world");
    println!("{a}");
};
```

What happens when we call:

```
fs();
fs();
println!("{a}");
```

FIN