Meetup:

Introduction to Rust

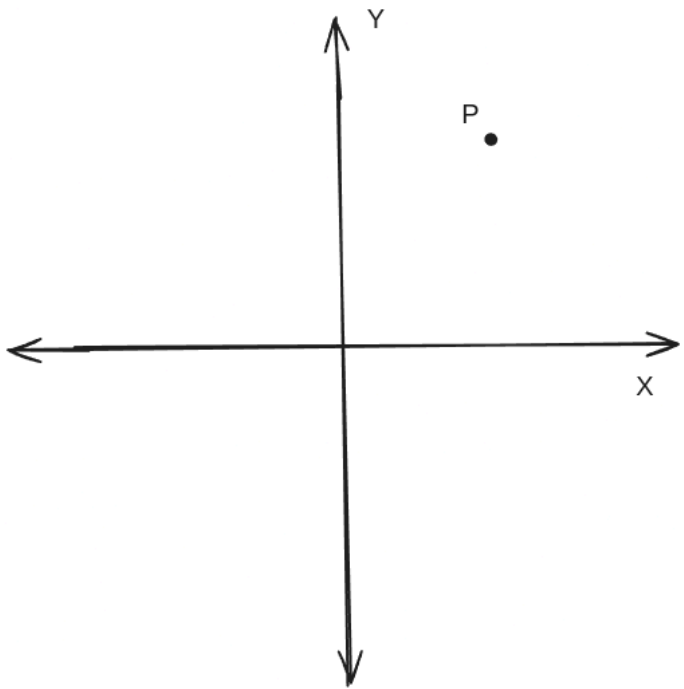
Discover the power of safe and efficient programming with Rust

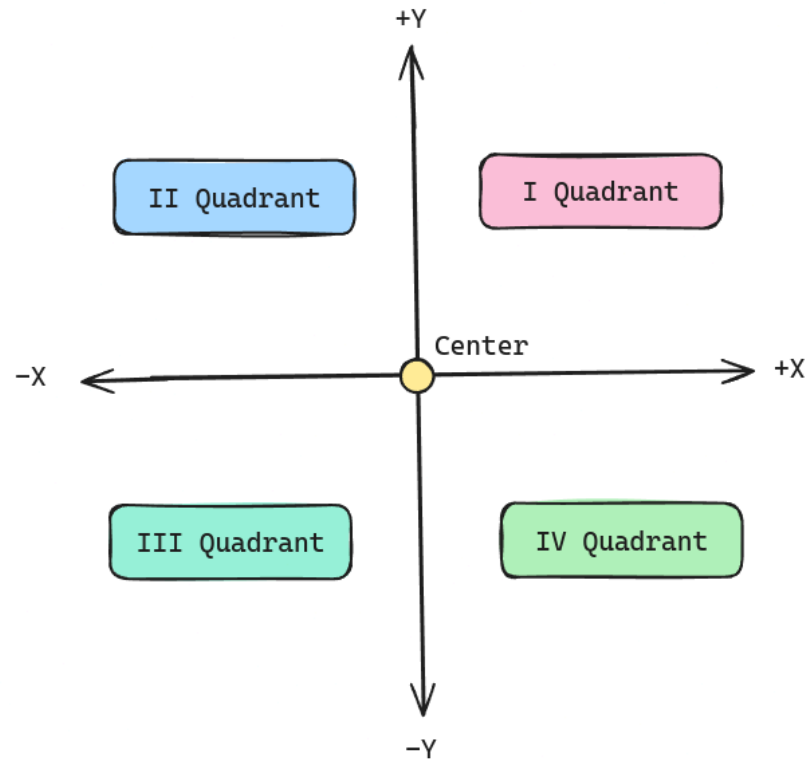
Press Space for next page →

Compiler for Modern Developers

- Cargo toolset (e.g. fmt, clippy, code analyzer)
- **Compiler's feedback**
- **Incremental compilation**

Point on a Plane



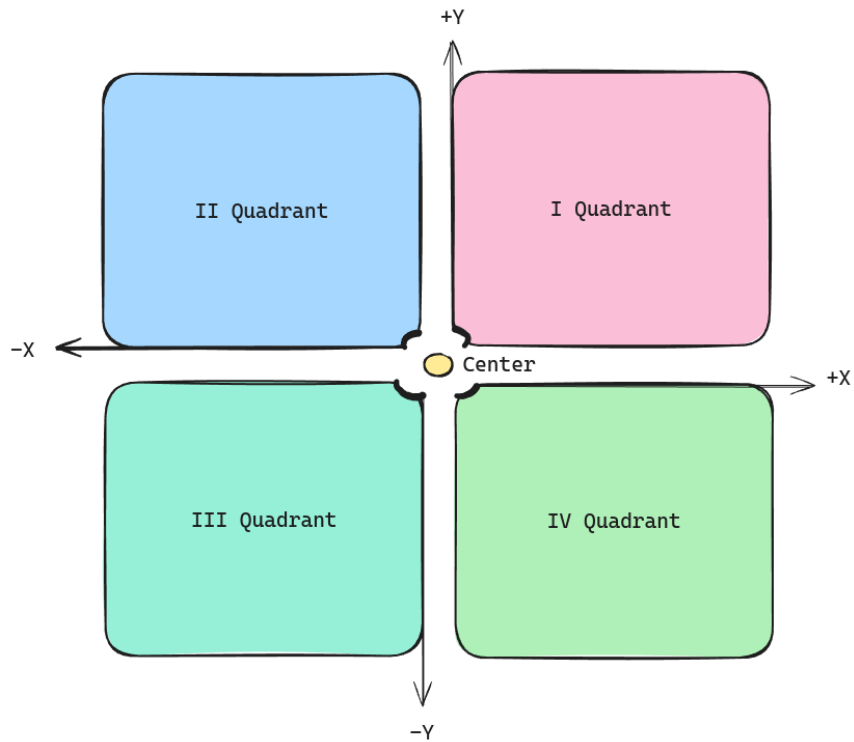


Algebraic Typing

Structs

```
struct Point {
    x: f32,
    y: f32
}
```

Algebraic Typing can be seen as sets.



Immutability by Default

Decide conscientemente cuando tus variables sean capaces de mutar

```
fn main() {
    let p = Point {x: 3.0, y: 4.2};
    p.x = 13.0;
    // Error: cannot assign to `p.x`,
    // as `p` is not declared as mutable
}
```

Enumerators and Traits

```
enum QuadrantNum {
    CENTER,
    I,
    II,
    III,
    IV
}

trait Quadrant {
    fn get_quadrant(self) → QuadrantNum;
}
```

Implementing Quadrant Trait

```
impl Quadrant for Point {
    fn get_quadrant(self) → QuadrantNum {
        if self.x > 0.0 && self.y ⩾ 0.0 {
            QuadrantNum::I
        } else if self.x ⩽ 0.0 && self.y > 0.0 {
            QuadrantNum::II
        } else if self.x < 0.0 && self.y ⩽ 0.0 {
            QuadrantNum::III
        } else if self.x ⩾ 0.0 && self.y < 0.0 {
            QuadrantNum::IV
        } else {
            QuadrantNum::CENTER
        }
    }
}
```

Pattern Matching

`match` keyword

```
impl Point {
    fn show_quad(self) {
        match self.get_quadrant() {
            QuadrantNum::I ⇒ println!("Quadrant I"),
            QuadrantNum::II ⇒ println!("Quadrant II"),
            QuadrantNum::III ⇒ println!("Quadrant III"),
            QuadrantNum::IV ⇒ println!("Quadrant IV"),
            QuadrantNum::CENTER ⇒ println!("It's the origin"),
        }
    }
}
```

Owning and Security

(Ownership & Borrow Checking)

fighting the borrow checker

```
fn change(s: &mut String) {
    s.push_str(", world");
}

fn main() {
    let mut s = String::from("hello");
    change(&mut s);

    change(&mut s);
}
```

Heap and Stack

```
fn main() {
    let arr: [i32; 5] = [1, 2, 3, 4, 5];
    let v = vec![1, 2, 3, 4, 5];

    println!("Array: {:?}", arr);
    println!("Vector: {:?}", v);
}
```

Compilation Time Programming or Metaprogramming

```
fn main() {
    todo!() // macros
}
```

Poem and OpenAPI

Hello World in a Web Server

```
struct Api;

#[OpenApi]
impl Api {
    #[oai(path = "/hello", method = "get")]
    async fn index(&self, name: Query<Option<String>>) -> PlainText<String> {
        match name.0 {
            Some(name) => PlainText(format!("hello, {name}!")),
            None => PlainText("hello!".to_string()),
        }
    }
}
```

Chat with `async-openai`

```
let request = CreateChatCompletionRequestArgs::default()
    .max_tokens(1024u16)
    .model(model_name)
    .messages([
        ChatCompletionRequestSystemMessageArgs::default()
            .content("You are a helpful assistant.")
            .build()?
            .into(),
        ChatCompletionRequestUserMessageArgs::default()
            .content("Hello there!")
            .build()?
            .into(),
    ])
    .build()?;

println!("{}", serde_json::to_string(&request).unwrap());

let response = client.chat().create(request).await?;
```

Gracias!