Prelude

Prelude

El prelude es la lista de “cosas” que rust importa automáticamente en cada programa.
Se mantiene lo más pequeño posible y hace foco en los traits que se usan en la mayoría de los programas.

Prelude

```rust
// https://doc.rust-lang.org/std/prelude/index.html
mod tp1;
fn main() {
    let o:Option<i32> = None;
    let r: Result<i32,ParseError>;
    let v: Vec<i32>;
    let s: String;
    let algo: HashMap<i32, i32> = HashMap::new();
}
```

Archivos

Archivos

El struct File representa un archivo abierto (envuelve un file descriptor) y da acceso de lectura y/o escritura al archivo subyacente.

Dado que pueden salir muchas cosas mal en la E/S de archivos, todos los métodos de archivo devuelven el tipo io::Result<T>, que es un alias para Result<T, io::Error>.

Esto hace que la falla de todas las operaciones de E/S sean explícitas.

Archivos: open

```rust
use std::fs::File;
use std::io::prelude::*;
use std::path::Path;
fn main() {
    let path = "src/archivo1.txt";
    // Abre el archivo del path en modo lectura, retorna `io::Result<File>`
    let mut archivo:File = match File::open(path) {
        Err(e) => panic!("No se pudo abrir por:  {}", e),
        Ok(archivo) => {archivo}
    };
    // Lee el contenido en un string, retorna `io::Result<usize>`
    let mut s = String::new();
    match archivo.read_to_string(&mut s) {
        Err(e) => panic!("No se puede leer por:  {}", e),
        Ok(_) => print!("contiene: \n{}", s),
    }
    // cuando termina el scope el archivo se cierra
}
```

Archivos: create

```rust
use std::fs::File;
use std::io::prelude::*;
use std::path::Path;
fn main() {
    let path = "src/archivo2.txt";
    // Abre el archivo en modo solo escritura, retorna `io::Result<File>`
    let mut archivo = match File::create(path) {
        Err(e) => panic!("No se puede crear porque: {}", e),
        Ok(archivo) => archivo,
    };

  // Escribe un string al archivo, retorna `io::Result<()>`
    match archivo.write_all("Limpieza total".as_bytes()) {
        Err(e) => panic!("No puede escrinbir porque: {}", e),
        Ok(_) => println!("Escribió correctamente en: {}" ,path),
    }
}
```

Archivos: read_lines

```rust
use std::fs::File;
use std::io::{ self, BufRead, BufReader };
fn main() {
    let path = "src/archivo1.txt";
    let archivo = File::open(path).unwrap();
    let mut lineas = BufReader::new(archivo).lines();
    for linea in lineas {
        println!("{:#?}", linea);
    }
}
```

Archivos: read_lines

```rust
use std::fs::File;
use std::io::{ self, BufRead, BufReader };
fn main() {
    let archivo = File::open("src/archivo1.txt".to_string()).unwrap();

    let mut lineas = io::BufReader::new(archivo).lines();
    for linea in lineas {
        println!("{:#?}", linea);
    }
}
```

Archivos con serde

Serde es un framework para serializar y deserializar structs de Rust de manera eficiente y genérica.

El ecosistema Serde consiste en estructuras de datos que saben cómo serializarse y deserializarse a sí mismos junto con formatos de datos que saben cómo serializar y deserializar otras cosas. Serde proporciona la capa por la cual estos dos grupos interactúan entre sí, lo que permite serializar y deserializar cualquier estructura de datos compatible utilizando cualquier formato de datos compatible.

Algunos formatos: JSON, YAML, TOML, Pickle, BSON,

Archivos con serde

```rust
use serde::{Serialize, Deserialize};
#[derive(Serialize, Deserialize, Debug)]
struct Punto {
    x: i32,
    y: i32,
}
fn main() {
    let p = Punto { x: 1, y: 2 };
    // Convierte el punto a un String con formato JSON.
    let punto_serializado = serde_json::to_string(&p).unwrap();
    println!("serialized = {}", punto_serializado);

    // Convierte el String con formato JSON de vuelta a un Punto.
    let p_s: Punto = serde_json::from_str(&punto_serializado).unwrap();
    println!("deserialized = {:?}", p_s);
}
```

Archivos con serde

```
fn main() {
    let p = Punto { x: 1, y: 2 };
    let punto_serializado = serde_json::to_string(&p).unwrap();
    let mut f = File::create("src/archivo_puntos.json").unwrap();
    f.write_all(&punto_serializado.as_bytes());

    let mut f_o: File = File::open("src/archivo_puntos.json").unwrap();
    let mut buf = String::new();
    f_o.read_to_string(&mut buf);
    let p1: Punto = serde_json::from_str(&buf).unwrap();
    println!("{:?}", p1);
}
```

Archivos con serde

```
fn main() {
    let mut v = Vec::new();
    let p = Punto { x: 1, y: 2 };
     v.push(&p);
    let p = Punto { x: 5, y: 12 };
    v.push(&p);
    let v_s = serde_json::to_string(&v).unwrap();
    let mut f = File::create("src/archivo_puntos.json").unwrap();
    f.write_all(&v_s.as_bytes());

    let mut f_o: File = File::open("src/archivo_puntos.json").unwrap();
    let mut buf = String::new();
    f_o.read_to_string(&mut buf);
    let mut v1: Vec<Punto> = serde_json::from_str(&buf).unwrap();
    println!("{:?}", v1);
}
```