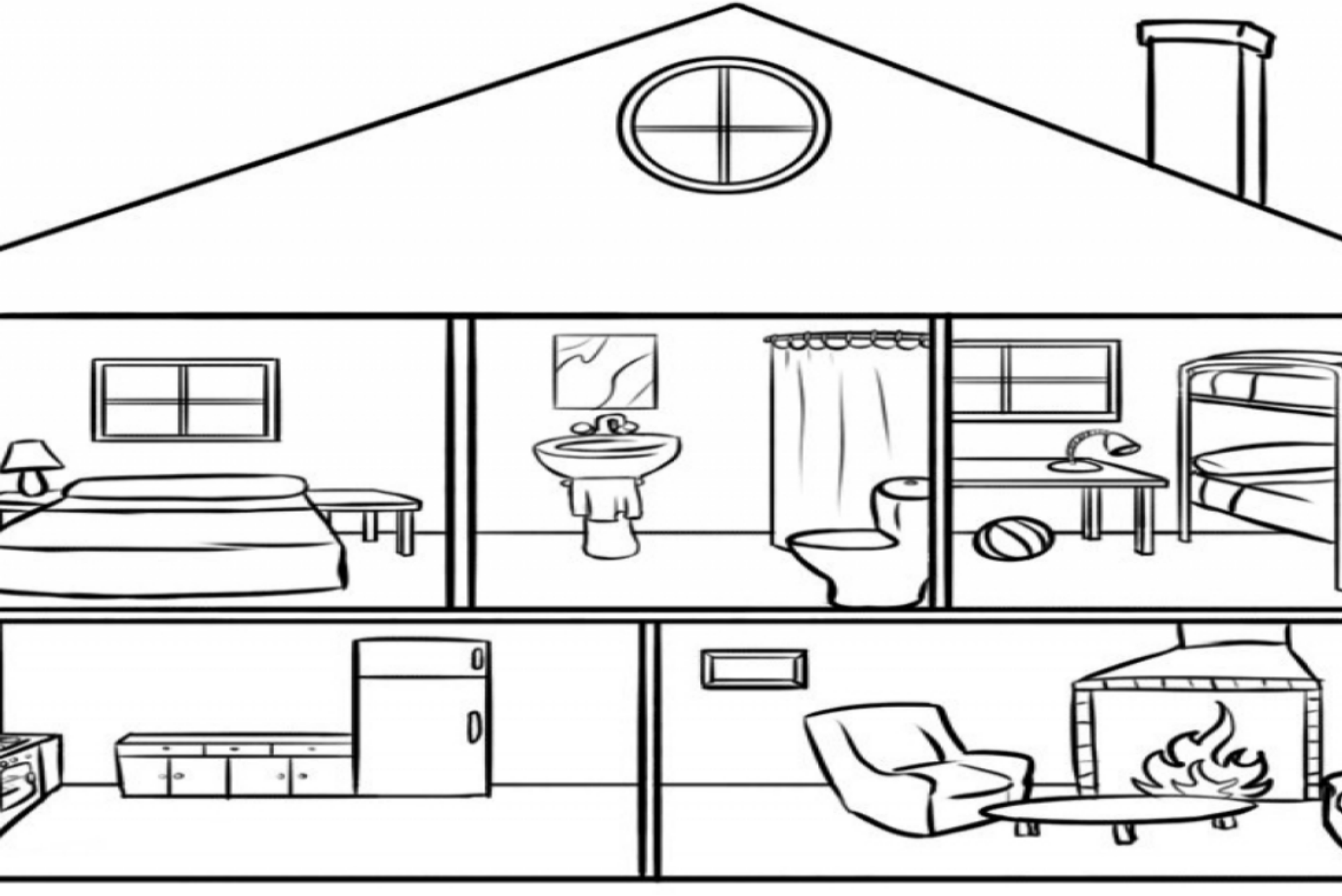
STRUCTS Y ENUMS

OBJETIVOS:

- Saber aplicar y utilizar los structs
- Diferenciar entre el uso de una tupla y un struct
- Conocer los diferentes tipos de structs
- Saber aplicar y utilizar los enums
- Usar el patrón de coincidencias de match para manejar enumeraciones

STRUCTS



Escribiendo el código de la casa usando variables:

ESTRUCTURA DE UN STRUCT

- Palabra clave **struct**
- Nombre del struct:
- Las variables que lo contienen

Tipo de estructura: Named fields

```
struct MyStruct {
    foo: i32,
    bar: f32,
}
```

Ejemplo de la casa utilizando structs:

Calculando el área de un rectángulo usando variables:

Calculando el área de un rectángulo usando tuplas:

Calculando el área de un rectángulo usando structs: Agregando significado

MÉTODOS

Son similares a las funciones:

- Son declarados con la palabra clave **fn**
- Tienen parámetros
- Tienen valor de retorno
- Contienen código que se ejecuta cuando son llamados desde otro lugar.

MÉTODOS

- Se definen dentro del contexto de un struct (Enums o Traits)
- Su primer parámetro es siempre **self**

Calculo del área de un rectángulo con impl:

Funciones asociativas:

TUPLE STRUCT

- Tienen el significado agregado en el nombre
- Solo tienen los tipos de los campos

Ejemplo de Tuple struct:

UNIT STRUCT

- No tienen ningún campo
- Similar al valor unitario ()
- Son utiles cuando necesitamos implementar un trait de un tipo

ENUMS

Nos permiten definir un tipo al enumerar sus posibles valores.

Son más similares a los tipos de datos algebraicos en lenguajes funcionales como:

- OCalm
- Haskell

Ejemplo de enum almacenando los datos en un struct:

Dirección IP utilizando solo enumeración:

Cada variante puede tener diferentes tipos y cantidades de datos asociados: Ejemplo

Ejemplo de enums con impl:

OPTION

- Rust no tiene null
- Rust tiene un enum (Option) cuyo concepto es si el valor está presente o ausente
- El compilador checa si se está manejando todos los posibles casos.
- Esta definida en la libreria estandar

```
enum Option<T>{
        Some(T),
        None,
}
```

MATCH

- Permite comparar un valor contra una serie de patrones y ejecutar código en cada patrón que coincide
- En Rust los matches son exhaustivos: Debemos de agotar todas las posibilidades para que el código sea válido.

El match se conforma de dos partes:

- El patrón
- El código

Ejemplo de la casa usando Option y Match:

Ejemplo de continentes con Match :

El _ Placeholder:

- Patrón para no enumerar todos los valores posibles.

Ejemplo de tallas de camisas.

Sintaxis **if let**

- Permite combinar **if** y **let** de una forma más sencilla para manejar valores que coincidan con un patrón mientras ignoramos el resto.
- Menos escritura
- menos sangría

Sin embargo, pierdes la comprobación exhaustiva que impone el match

Ejemplo de color favorito usando if let:

Ejercicios:

Crear un programa que permita crear diferentes tipos de libro, cada libro debe de tener el nombre y apellido del autor, el nombre del libro y una o varias secciones

Ejemplo de secciones:

- Ciencia
  Ficción
- Clásicos

Preguntas

Referencias:

- The Rust Team, (2018), *The Rust Programming Language (Second Edition)*
- Blandy Jim, Orendorff Jason, (2017), *Programming Rust*, O'Reilly Media, Inc
- Balbaert Ivo, (2017), *Rust essentials*

¡GRACIAS!