Enums With Data Type

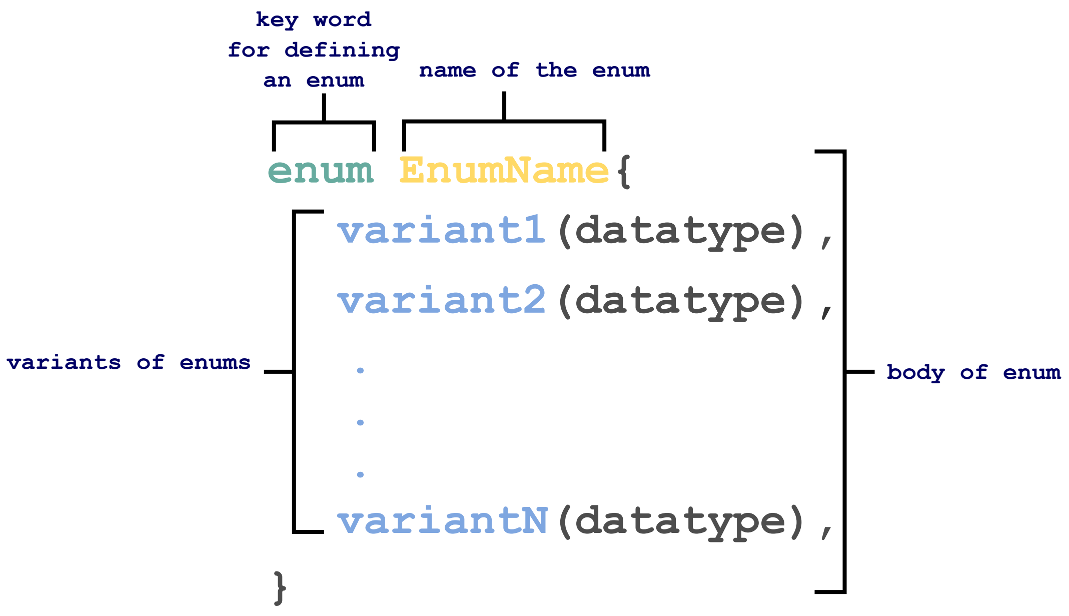
This lesson tells you how to make an enum construct by adding a data type.

We'll cover the following

By default, the Rust compiler infers the data type for all variants of an enum. However, it is possible to use different data types for different variants of an enum.

Syntax

The data type can be added to each variant enclosed within round brackets `()`.



Defining enum variants with data type

Example

The following example makes an `enum` `KnightMove` having two variants `Horizontal` and `Vertical` both of type `String`.

```
// make this `enum` printable with `fmt::Debug`.
#[derive(Debug)]
enum KnightMove{
   Horizontal(String), Vertical(String)
}
fn main() {
   // invoke an enum
   let horizontal_move = KnightMove::Horizontal("Left".to_string());
   let vertical_move = KnightMove::Vertical("Down".to_string());
   // print enum
   println!("Move 1: {:?}", horizontal_move);
   println!("Movw 2: {:?}", vertical_move);
}
```

Explanation

- `main` **Function**

  The body of the main function is defined from **line 6 to line 13**.

  - On **line 8** and **line 9**, enum is initialized and the values are saved in `horizontal_move` and `vertical_move`. Since the variants are declared as String, we are creating a String object from a string literal.
  - On **line 11** and **line 12**, the values of `enum` are printed.

- On **line 2**, `#[derive(Debug)]` is declared which helps to print the values of the enum.

- `enum`

  - On **line 3**, `enum` `KnightMoves` is defined.
  - On **line 4**, **variants** of enum `Horizontal` and `Vertical` both of type `String` are defined.

```
#[derive(Debug)]
enum KnightMove{
   Horizontal(String),Vertical(String)
}
fn main() {
   let horizontal_move = KnightMove::Horizontal("Left".to_string());
   let vertical_move = KnightMove::Vertical("Down".to_string());
   println!("{:?}",horizontal_move);
   println!("{:?}",vertical_move);
}
```

Output:

```
#[derive(Debug)]
enum KnightMove{
   Horizontal(String),Vertical(String)
}
fn main() {
   let horizontal_move = KnightMove::Horizontal("Left".to_string());
   let vertical_move = KnightMove::Vertical("Down".to_string());
   println!("{:?}",horizontal_move);
   println!("{:?}",vertical_move);
}
```

Output:

```
#[derive(Debug)]
enum KnightMove{
   Horizontal(String),Vertical(String)
}
fn main() {
   let horizontal_move = KnightMove::Horizontal("Left".to_string());
   let vertical_move = KnightMove::Vertical("Down".to_string());
   println!("{:?}",horizontal_move);
   println!("{:?}",vertical_move);
}
```

Output:

```
#[derive(Debug)]
enum KnightMove{
   Horizontal(String),Vertical(String)
}
fn main() {
   let horizontal_move = KnightMove::Horizontal("Left".to_string());
   let vertical_move = KnightMove::Vertical("Down".to_string());
   println!("{:?}",horizontal_move);
   println!("{:?}",vertical_move);
}
```

Output:

```
Horizontal("Left")
```

```
#[derive(Debug)]
enum KnightMove{
   Horizontal(String),Vertical(String)
}
fn main() {
   let horizontal_move = KnightMove::Horizontal("Left".to_string());
   let vertical_move = KnightMove::Vertical("Down".to_string());
   println!("{:?}",horizontal_move);
   println!("{:?}",vertical_move);
}
```

Output:

```
Horizontal("Left")
Vertical("Down")
```

5 of 6

```
#[derive(Debug)]
enum KnightMove{
   Horizontal(String),Vertical(String)
}
fn main() {
   let horizontal_move = KnightMove::Horizontal("Left".to_string());
   let vertical_move = KnightMove::Vertical("Down".to_string());
   println!("{:?}",horizontal_move);
   println!("{:?}",vertical_move);
} end of program code
```

Output:

```
Horizontal("Left")
Vertical("Down")
```

6 of 6

Now that you have learned the basics of enums, let’s learn about methods in enums in the next lesson.