Declarative Macros ❙ Overview

Declarative Macros

- A form of metaprogramming (code that writes code)
- Hygienic:
  - Unable to emit invalid code
  - Data cannot “leak” in to (or out of) a macro
    - Macros *cannot* capture information like closures
    - All names / bindings / variables must be provided by the caller
- Uses macro-specific pattern matching to emit code
- Invoked using an exclamation point: **macro_name!()**

Invoking a Macro

```
your_macro_name!();
your_macro_name![];
your_macro_name!{}
```

Valid Positions

- Macros can only be used in specific parts of Rust code:
  - Expressions & Statements
  - Patterns
  - Types
  - Items & Associated Items
  - macro_rules transcribers
  - External blocks

Expression & Statement Position

```rust
// Expressions
let nums = vec![1, 2, 3];

match vec![1, 2, 3].as_slice() {
    _ => format!("hello"),
}

// Statements
println!("Hello!");

dbg!(9_i64.pow(2));
```

Pattern Position

```rust
macro_rules! pat {
    ($i:ident) => (Some($i))
}

// Patterns
if let pat!(x) = Some(1) {
    assert_eq!(x, 1);
}

match Some(1) {
    pat!(x) => (),
    _ => (),
}
```

Type Position

```rust
macro_rules! Tuple {
    { $A:ty, $B:ty } => { ($A, $B) };
}

// Types
type N2 = Tuple!(i32, i32);

let nums: Tuple!(i32, char) = (1, 'a');
```

Item Position

```rust
macro_rules! constant {
    ($name:ident) => { const $name: &'static str = "Jayson"; }
}
macro_rules! newtype {
    ($name:ident, $typ:ty) => { struct $name($typ); }
}

// Items
constant!(NAME);
assert_eq!(NAME, "Jayson");

newtype!(DemoStruct, usize);
let demo = DemoStruct(5);
```

Associated Item Position

```rust
macro_rules! msg {
    ($msg:literal) => {
        pub fn msg() {
            println!("{}", $msg);
        }
    };
}
struct Demo;
// Associated item
impl Demo {
    msg!("demo struct");
}
```

macro_rules Transcribers

```rust
// macro_rules transcribers
macro_rules! demo {
    () => {
        println!("{}",
            format!("demo{}", '!')
        );
    };
}
demo!();
```

Recap

- Macros are a form of metaprogramming
- Invoked using an exclamation point (!)
  - Invocation can be done with parentheses (), curly braces {}, or square braces []
- Are valid in many (but not all) positions
- Macros can invoke other macros, including recursive invocation