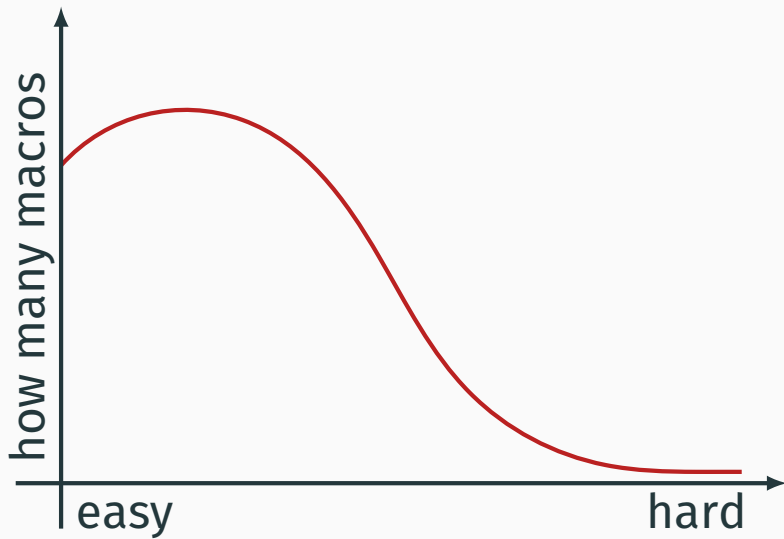


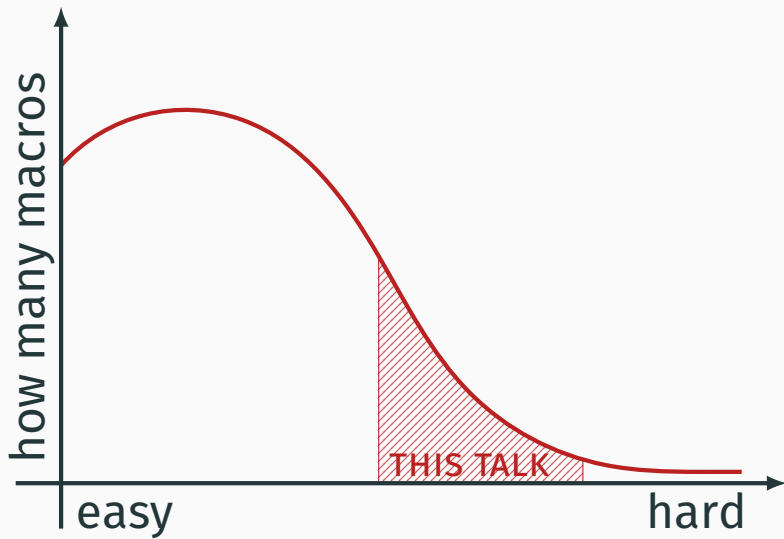


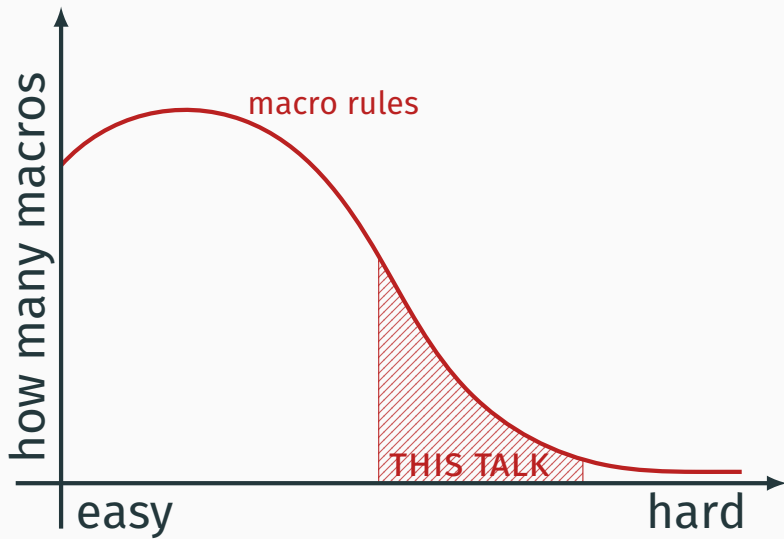


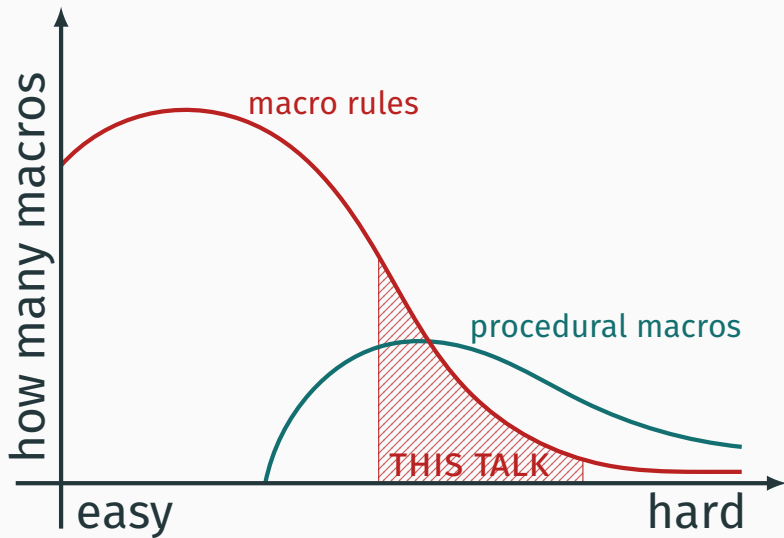
# RUST AS TT SEES IT

dtolnay









```
let response = json!({  
    "status": "error",  
    "retry": false,  
    "message": "File not found."  
});
```



```
let response = {  
  let mut map = HashMap::new();  
  map.insert("status", String("error"));  
  map.insert("retry", Boolean(false));  
  map.insert("message", String("File not f..  
  map  
};
```

```
let response = "{  
  \"status\": \"error\",  
  \"retry\": false,  
  \"message\": \"File not found.\"  
}";
```

```
let response = r#{  
  "status": "error",  
  "retry": false,  
  "message": "File not found."  
}";
```

```
let message = /* ... */;
```

```
let response = json!({  
    "status": "error",  
    "retry": false,  
    "message": message  
});
```

```
let message = /* ... */;
```

```
let response = json!({  
  "status": "error",  
  "retry": false,  
  "message": message  
});
```

```
let message = /* ... */;
```

```
let response = json!({  
    "status": "error",  
    "retry": false,  
    "message": message  
});
```

```
let response = json!({  
    "status": "error",  
    "retry": i < max_retries,  
    "message": /* ... */  
});
```

```
let response = json!({  
  "status": "error",  
  "retry": i < max_retries,  
  "message": /* ... */  
});
```



# SEQUENCE POINT


## Covered:

- Invoking a macro — `m!(...)`

## Open questions:

- What the heck is TT?
- Differences from C preprocessor macros
- Defining a macro

```
macro_rules! json {  
    (...) => { ... }  
}
```

```
macro_rules! json {  
    (  ) => { ... }  
}
```

```
macro_rules! json {  
    (...) => { ... }  
}
```

```
macro_rules! json {  
    (... ) => { ... }  
  
    (... ) => { ... }  
}
```

```
macro_rules! demo {  
    (step1) => { demo!(step2) }  
    (step2) => { "done" }  
}
```

```
let output = demo!(step1);
```

```
macro_rules! demo {  
    (step1) => { demo!(step2) }  
    (step2) => { "done" }  
}
```

```
let output = demo!(step1);
```

```
macro_rules! demo {  
    (step1) => { demo!(step2) }  
    (step2) => { "done" }  
}
```

```
let output = demo!(step1);
```



```
macro_rules! demo {  
    (step1) => { demo!(step2) }  
    (step2) => { "done" }  
}
```

```
let output = demo!(step1);
```

```
macro_rules! demo {  
    (step1) => { demo!(step2) }  
    (step2) => { "done" }  
}
```

```
let output = demo!(step1);
```

```
macro_rules! demo {  
    (step1) => { demo!(step2) }  
    (step2) => { "done" }  
}
```

```
let output = demo!(step1);
```

```
macro_rules! demo {  
    (step1) => { demo!(step2) }  
    (step2) => { "done" }  
}
```

```
let output = demo!(step1);
```

```
macro_rules! demo {  
    (step1) => { demo!(step2) }  
    (step2) => { "done" }  
}
```

```
let output = demo!(step1);
```

```
macro_rules! demo {  
    (step1) => { demo!(step2) }  
    (step2) => { "done" }  
}
```

```
let output = demo!(step1);
```

```
macro_rules! demo {  
    (step1) => { demo!(step2) }  
    (step2) => { "done" }  
}
```

```
let output = demo!(step1);
```

```
macro_rules! demo {  
    (step1) => { demo!(step2) }  
    (step2) => { "done" }  
}
```

```
let output = demo!(step1);
```



```
macro_rules! demo {  
    (step1) => { demo!(step2) }  
    (step2) => { "done" }  
}
```

```
let output = demo!(step1);
```

```
macro_rules! status_codes {  
    ($($k:expr => $v:expr ,)* ) => {  
        /* ... */  
    }  
}
```

```
macro_rules! status_codes {  
    ($($k:expr => $v:expr ,)* ) => {  
        /* ... */  
    }  
}
```

```
macro_rules! status_codes {  
    ($($k:expr => $v:expr ,)* ) => {  
        /* ... */  
    }  
}
```

```
macro_rules! status_codes {  
    ($($k:expr => $v:expr ,)* ) => {  
        /* ... */  
    }  
}
```

```
macro_rules! status_codes {  
    ($($k:expr => $v:expr ,)* ) => {  
        /* ... */  
    }  
}
```

```
macro_rules! status_codes {  
    ($($k:expr => $v:expr ,)*) => {  
        /* ... */  
    }  
}
```

```
status_codes! {  
    404 => "Not Found",  
    408 => "Request Timeout",  
}
```

```
macro_rules! status_codes {  
    ($($k:expr => $v:expr ,)* ) => {  
        /* ... */  
    }  
}
```

```
status_codes! {  
    404 => "Not Found",  
    408 => "Request Timeout",  
}
```



```
macro_rules! status_codes {  
    ($($k:expr => $v:expr ,)*) => {  
        /* ... */  
    }  
}
```

```
status_codes! {  
    404 => "Not Found",  
    408 => "Request Timeout",  
}
```

```
macro_rules! status_codes {  
    ($($k:expr => $v:expr ,)*) => {  
        /* ... */  
    }  
}
```

```
status_codes! {  
    404 => "Not Found",  
    408 => "Request Timeout",  
}
```

```
macro_rules! status_codes {  
    ($($k:expr => $v:expr ,)*) => {  
        /* ... */  
    }  
}
```

```
status_codes! {  
    404 => "Not Found",  
    408 => "Request Timeout",  
}
```

```
macro_rules! status_codes {  
    ($($k:expr => $v:expr ,)* ) => {  
        /* ... */  
    }  
}
```

```
status_codes! {  
    404 => "Not Found",  
    408 => "Request Timeout",  
}
```

```
macro_rules! status_codes {  
    ($($k:expr => $v:expr ,)*) => {  
        /* ... */  
    }  
}
```

```
status_codes! {  
    404 => "Not Found",  
    408 => "Request Timeout",  
}
```

```
macro_rules! status_codes {  
    ($($k:expr => $v:expr ,)*) => {  
        /* ... */  
    }  
}
```

```
status_codes! {  
    404 => "Not Found",  
    408 => "Request Timeout",  
}
```

```
macro_rules! status_codes {  
    ($($k:expr => $v:expr ,)*) => {  
        /* ... */  
    }  
}
```

```
status_codes! {  
    404 => "Not Found",  
    408 => "Request Timeout",  
}
```

```
macro_rules! status_codes {  
    ($($k:expr => $v:expr ,)* ) => {  
        /* ... */  
    }  
}
```

```
status_codes! {  
    404 => "Not Found",  
    408 => "Request Timeout",  
}
```



```
macro_rules! status_codes {  
    ($($k:expr => $v:expr ,)*) => {  
        /* ... */  
    }  
}
```

```
status_codes! {  
    404 => "Not Found",  
    408 => "Request Timeout",  
}
```

```
macro_rules! status_codes {  
    ($($k:expr => $v:expr ,)* ) => {  
        /* ... */  
    }  
}
```

```
status_codes! {  
    404 => "Not Found",  
    408 => "Request Timeout",  
}
```

```
macro_rules! status_codes {  
    ($($k:expr => $v:expr ,)* ) => {  
        /* ... */  
    }  
}
```

```
status_codes! {  
    404 => "Not Found",  
    408 => "Request Timeout",  
}
```

# SEQUENCE POINT

## Covered:

- Macro rules
- Fragment variables — `$e:expr`
- Repetitions — `$(...)*`

## Open questions:

- What the heck is TT?
- Ambiguities?

```
let response = json!({  
    "status": "error",  
    "retry": false,  
    "message": "File not found."  
});
```

```
macro_rules! json {  
    ([ ... ]) => { ... }  
    ({ ... }) => { ... }  
    ($other:expr) => { ... }  
}
```

```
macro_rules! json {  
    ([ ... ]) => { ... }  
    ({ ... }) => { ... }  
    ($other:expr) => { ... }  
}
```

```
macro_rules! json {  
    ([ ... ]) => { ... }  
    ({ ... }) => { ... }  
    ($other:expr) => { ... }  
}
```



```
macro_rules! json {  
    ([ ... ]) => { ... }  
    ({ ... }) => { ... }  
    ($other:expr) => { ... }  
}
```

```
macro_rules! json {  
    ([ ... ]) => { ... }  
    ({ ... }) => { ... }  
    ($other:expr) => { ... }  
}
```

```
macro_rules! json {  
    ([ ... ]) => { ... }  
    ({ ... }) => { ... }  
    ($other:expr) => { ... }  
}
```

```
json!([1, {"k": "v"}, i < retries])
```

```
json!([1, {"k": "v"}, i < retries])
```

```
json!([1, {"k": "v"}, i < retries])
```

```
json!([1, {"k": "v"}, i < retries])
```

```
json!([1, {"k": "v"}, i < retries])
```

```
macro_rules! json {  
    ([ ??? ]) => { ... }  
    // other rules  
}
```



```
json!([1, {"k": "v"}, i < retries])
```

```
macro_rules! json {  
    ([ $($e:expr ,)* ]) => { ... }  
    // other rules  
}
```

```
json!([1, {"k": "v"}, i < retries])
```

```
macro_rules! json {  
    ([ $( $e:expr , )* ]) => { ... }  
    // other rules  
}
```

```
json!([1, {"k": "v"}, i < retries])
```

```
macro_rules! json {  
    ([ $(e:expr ,)* ]) => { ... }  
    // other rules  
}
```

```
json!([1, {"k": "v"}, i < retries])
```

```
macro_rules! json {  
    ([ $( [...] OR {...} OR $expr, )* ]) => {  
        ...  
    }  
    // other rules  
}
```

```
json!([1, {"k": "v"}, i < retries])
```

```
macro_rules! json {  
    ([ $([..] OR {...} OR $expr,)* ]) => {  
        ...  
    }  
    // other rules  
}
```

```
json!([1, {"k": "v"}, i < retries])
```

```
macro_rules! json {  
    ([ $( [...] ,)* ]) => { ... }  
    ([ $( {...} ,)* ]) => { ... }  
    ([ $( $e:expr ,)* ]) => { ... }  
    // other rules  
}
```

```
json!([1, {"k": "v"}, i < retries])
```

```
macro_rules! json {  
    ([ $( [ ... ] , ) * ]) => { ... }  
    ([ $( { ... } , ) * ]) => { ... }  
    ([ $( $e:expr , ) * ]) => { ... }  
    // other rules  
}
```

```
json!([1, {"k": "v"}, i < retries])
```

```
macro_rules! json {  
    ([ $( [...] ,)* ]) => { ... }  
    ([ $( {...} ,)* ]) => { ... }  
    ([ $( $e:expr ,)* ]) => { ... }  
    // other rules  
}
```



```
json!([1, {"k": "v"}, i < retries])
```

```
macro_rules! json {  
    ([ $( [...] ,)* ]) => { ... }  
    ([ $( {...} ,)* ]) => { ... }  
    ([ $( $e:expr ,)* ]) => { ... }  
    // other rules  
}
```

```
json!([1, {"k": "v"}, i < retries])
```

```
macro_rules! json {  
    ([ $( [...] ,)* ]) => { ... }  
    ([ $( {...} ,)* ]) => { ... }  
    ([ $( $e:expr ,)* ]) => { ... }  
    // other rules  
}
```

WHAT IS GOING WRONG?

# WHAT IS GOING WRONG?

- Macro rules optimize for Rust-like grammars
- Decision:
  - Make the input more like Rust, or
  - Power through

```
let response = json_map!(  
  "status" => "error",  
  "cause" => json::NULL,  
  "backtrace" => json_array!(  
    "failure::begin_unwind",  
    "option::Option<T>::unwrap",  
    "request::Request<S>::create"  
  ),  
);
```

```
let response = json_map!(  
  "status" => "error",  
  "cause" => json::NULL,  
  "backtrace" => json_array!(  
    "failure::begin_unwind",  
    "option::Option<T>::unwrap",  
    "request::Request<S>::create"  
  ),  
);
```

```
let response = json_map!(  
  "status" => "error",  
  "cause" => json::NULL,  
  "backtrace" => json_array!(  
    "failure::begin_unwind",  
    "option::Option<T>::unwrap",  
    "request::Request<S>::create"  
  ),  
);
```

```
let response = json_map!(  
  "status" => "error",  
  "cause" => json::NULL,  
  "backtrace" => json_array!(  
    "failure::begin_unwind",  
    "option::Option<T>::unwrap",  
    "request::Request<S>::create"  
  ),  
);
```



```
let response = json_map!(  
  "status" => "error",  
  "cause" => json::NULL,  
  "backtrace" => json_array!(  
    "failure::begin_unwind",  
    "option::Option<T>::unwrap",  
    "request::Request<S>::create"  
  ),  
);
```

```
let response = json_map!(  
  "status" => "error",  
  "cause" => json::NULL,  
  "backtrace" => json_array!(  
    "failure::begin_unwind",  
    "option::Option<T>::unwrap",  
    "request::Request<S>::create"  
  ),  
);
```

```
let response = json!({  
  "status": "error",  
  "cause": null,  
  "backtrace": [  
    "failure::begin_unwind",  
    "option::Option<T>::unwrap",  
    "request::Request<S>::create"  
  ]  
});
```

```
json!([1, {"k": "v"}, i < retries])
```

```
macro_rules! json {  
    ([ ??? ]) => { ... }  
    // other rules  
}
```

```
json!([1, {"k": "v"}, i < retries])
```

```
macro_rules! json {  
    ([ $(content:tt)* ]) => { ... }  
    // other rules  
}
```

```
json!([1, {"k": "v"}, i < retries])
```

```
macro_rules! json {  
    ([ $($content:tt)* ]) => { ... }  
    // other rules  
}
```

```
json!([1, {"k": "v"}, i < retries])
```

```
macro_rules! json {  
    ([ $(content:tt)* ]) => { ... }  
    // other rules  
}
```

```
json!([1, {"k": "v"}, i < retries])
```

```
macro_rules! json {  
    ([ $(content:tt)* ]) => { ... }  
    // other rules  
}
```



```
json!([1, {"k": "v"}, i < retries])
```


```
macro_rules! json {  
    ([ $($content:tt)* ]) => { ... }  
    // other rules  
}
```

```
json!([1, {"k": "v"}, i < retries])
```

1 2

```
macro_rules! json {  
    ([ $($content:tt)* ]) => { ... }  
    // other rules  
}
```


```
json!([1, {"k": "v"}, i < retries])
```



A diagram with three small brown brackets under the list elements. The first bracket is under '1' and labeled '1' below it. The second bracket is under '{ "k": "v" }' and labeled '2' below it. The third bracket is under 'i < retries' and labeled '3' below it.


```
macro_rules! json {  
    ([ $($content:tt)* ]) => { ... }  
    // other rules  
}
```

```
json!([1, {"k": "v"}, i < retries])
```




```
macro_rules! json {  
    ([ $($content:tt)* ]) => { ... }  
    // other rules  
}
```

```
json!([1, {"k": "v"}, i < retries])
```




```
macro_rules! json {  
    ([ $($content:tt)* ]) => { ... }  
    // other rules  
}
```

```
json!([1, {"k": "v"}, i < retries])
```




```
macro_rules! json {  
    ([ $($content:tt)* ]) => { ... }  
    // other rules  
}
```

```
json!([1, {"k": "v"}, i < retries])
```



```
macro_rules! json {  
    ([ $($content:tt)* ]) => { ... }  
    // other rules  
}
```

```
json!([1, {"k": "v"}, i < retries])
```



```
macro_rules! json {  
    ([ $($content:tt)* ]) => { ... }  
    // other rules  
}
```



```
macro_rules! json {  
    ([ $(content:tt)* ]) => {  
        validate_array!($($content)*);  
    }  
    // other rules  
}
```

```
macro_rules! json {  
    ([ $($content:tt)* ]) => {  
        validate_array!($($content)*);  
    }  
    // other rules  
}
```

```
macro_rules! json {  
    ([ $($content:tt)* ]) => {  
        validate_array!($($content)*);  
    }  
    // other rules  
}
```

```
macro_rules! json {  
    ([ $($content:tt)* ]) => {  
        validate_array!($($content)*);  
    }  
    // other rules  
}
```

```
macro_rules! json {  
    ([ $($content:tt)* ]) => {  
        validate_array!($($content)*);  
    }  
    // other rules  
}
```

```
macro_rules! validate_array {
```

```
macro_rules! validate_array {  
    () => {}  
}
```

```
macro_rules! validate_array {  
    () => {}  
    ([ $($c:tt)* ], $($rest:tt)*) => {  
        validate_array!($($c)*);  
        validate_array!($($rest)*);  
    }  
}
```



```
macro_rules! validate_array {  
    () => {}  
    ([ $($c:tt)* ], $($rest:tt)*) => {  
        validate_array!($($c)*);  
        validate_array!($($rest)*);  
    }  
}
```

```
macro_rules! validate_array {  
    () => {}  
    ([ $($c:tt)* ], $($rest:tt)*) => {  
        validate_array!($($c)*);  
        validate_array!($($rest)*);  
    }  
}
```

```
macro_rules! validate_array {  
    () => {}  
    ([ $($c:tt)* ], $($rest:tt)* ) => {  
        validate_array!($($c)*);  
        validate_array!($($rest)*);  
    }  
}
```

```
macro_rules! validate_array {  
    () => {}  
    ([ $($c:tt)* ], $($rest:tt)*) => {  
        validate_array!($($c)*);  
        validate_array!($($rest)*);  
    }  
}
```

```
macro_rules! validate_array {  
    () => {}  
    ([ $($c:tt)* ], $($rest:tt)*) => {  
        validate_array!($($c)*);  
        validate_array!($($rest)*);  
    }  
}
```

```
macro_rules! validate_array {  
    () => {}  
    ([ $($c:tt)* ], $($rest:tt)*) => {  
        validate_array!($($c)*);  
        validate_array!($($rest)*);  
    }  
}
```

```
macro_rules! validate_array {  
    () => {}  
    ([ $($c:tt)* ], $($rest:tt)*) => {  
        validate_array!($($c)*);  
        validate_array!($($rest)*);  
    }  
}
```

```
macro_rules! validate_array {  
    () => {}  
    ([ $($c:tt)* ], $($rest:tt)*) => {  
        validate_array!($($c)*);  
        validate_array!($($rest)*);  
    }  
}
```



```
macro_rules! validate_array {  
    () => {}  
    ([ $($c:tt)* ], $($rest:tt)*) => {  
        validate_array!($($c)*);  
        validate_array!($($rest)*);  
    }  
    ({ $($c:tt)* }, $($rest:tt)*) => {  
        ...  
    }  
}
```

```
macro_rules! validate_array {  
    () => {}  
    ([ $($c:tt)* ], $($rest:tt)*) => {  
        validate_array!($($c)*);  
        validate_array!($($rest)*);  
    }  
    ({ $($c:tt)* }, $($rest:tt)*) => {  
        ...  
    }  
}
```

```
macro_rules! validate_array {  
    () => {}  
    ([ $($c:tt)* ], $($rest:tt)*) => {  
        validate_array!($($c)*);  
        validate_array!($($rest)*);  
    }  
    ({ $($c:tt)* }, $($rest:tt)*) => {  
        ...  
    }  
    ($e:expr, $($rest:tt)*) => { ... }  
}
```

```

macro_rules! validate_array {
    () => {}
    ([ $($c:tt)* ], $($rest:tt)* ) => {
        validate_array!($($c)*);
        validate_array!($($rest)*);
    }
    ({ $($c:tt)* }, $($rest:tt)* ) => {
        ...
    }
    ($e:expr, $($rest:tt)* ) => { ... }
}

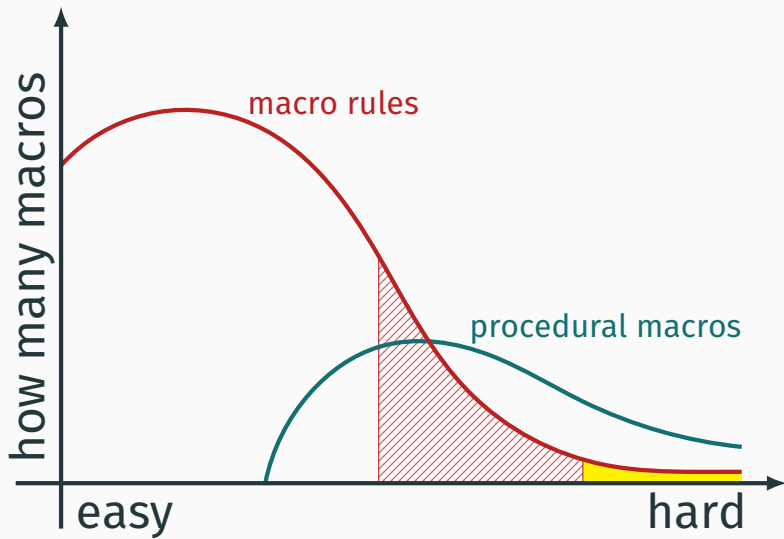
```

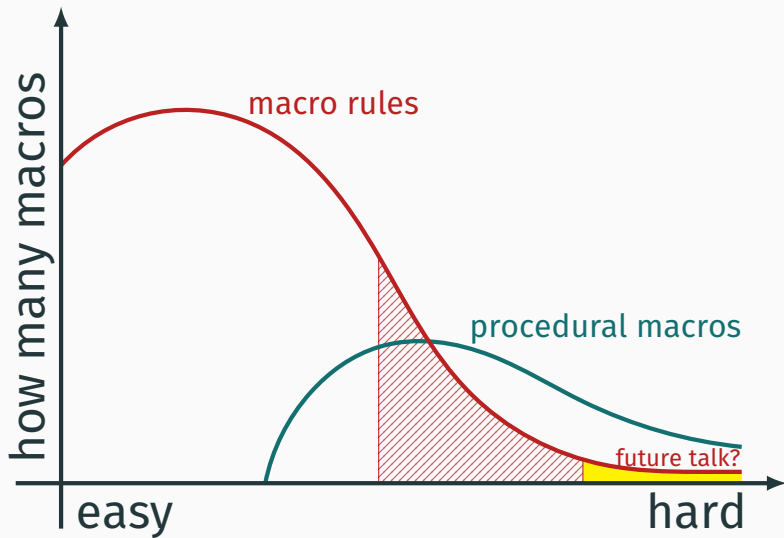
# The Little Book of Rust Macros

<https://danielkeep.github.io/tlborm/>

## 4. Patterns

- 4.1 Callbacks
- 4.2 TT Munchers
- 4.3 Internal Rules
- 4.4 Push-Down Accumulation
- 4.5 Repetition Replacement
- 4.6 Trailing Separators
- 4.7 TT Bundling





<https://github.com/dtolnay/tt-call>





QUESTIONS?



```
#![feature(trace_macros)]
trace_macros!(true);

macro_rules! demo {
    (step1) => { demo!(step2) }
    (step2) => { "done" }
}

fn main() {
    let output = demo!(step1);
}
```

```
--> src/main.rs:13:18
|
|      let output = demo!(step1);
|                      ^^^^^^^^^^^
= note: expanding 'demo!(step1)'
= note: to 'demo!(step2)'
= note: expanding 'demo!(step2)'
= note: to '"done"'
```

```
enum Value {  
    Null,  
    Bool(bool),  
    Number(Number),  
    String(String),  
    Array(Vec<Value>),  
    Object(Map<String, Value>),  
}
```

```
macro_rules! json {  
    ([ $($c:tt)* ]) => {  
        let mut array = Vec::new();  
        array_helper!(array, $($c)*);  
        Value::Array(array)  
    }  
    // other rules  
}
```

```
macro_rules! array_helper {  
    ($a:ident,) => {}  
    ($a:ident, [$($c:tt)*], $($r:tt)*) => {  
        $a.push(json!([$($c)*]));  
        array_helper!($a, $($r)*);  
    }  
    // other rules  
}
```



```
macro_rules! json {
    ([ $($c:tt)* ]) => {
        let mut array = Vec::new();
        array_helper!(array, $($c)*);
        Value::Array(array)
    }
    // other rules
}

macro_rules! array_helper {
    ($a:ident,) => {}
    ($a:ident, [ $($c:tt)* ], $($r:tt)* ) => {
        $a.push(json!([ $($c)* ]));
        array_helper!($a, $($r)*);
    }
    // other rules
}
```