

Crate | cached

■ cached

- ◆ Stores return value from a function
- ◆ Whenever the function is called, automatically returns the computed value
 - Large speedup when utilized with computation-heavy functions
- ◆ Supports:
 - TTL
 - Maximum cache size
 - Only store successful computations

Example - Basic

```
use cached::proc_macro::cached;
```

```
#[cached]
```

```
fn expensive(n: usize) -> usize {
```

```
    thread::sleep(Duration::from_millis(500));
```

```
    match n {
```

```
        1 => 1,
```

```
        2 => 2,
```

```
        _ => n,
```

```
    }
```

```
}
```

Example - Cache Limit

```
use cached::proc_macro::cached;
```

```
#[cached(size = 10)]
```

```
fn expensive(n: usize) -> usize {
```

```
    thread::sleep(Duration::from_millis(500));
```

```
    match n {
```

```
        1 => 1,
```

```
        2 => 2,
```

```
        _ => n,
```

```
    }
```

```
}
```

Example - TTL

```
use cached::proc_macro::cached;
```

```
#[cached(size = 50, time = 30)]
```

```
fn expensive(n: usize) -> usize {
```

```
    thread::sleep(Duration::from_millis(500));
```

```
    match n {
```

```
        1 => 1,
```

```
        2 => 2,
```

```
        _ => n,
```

```
    }
```

```
}
```

■ Example - Only Cache Success

```
#[cached(option = true)]
```

```
fn expensive_1(n: usize) -> Option<usize> { ...  
}
```

```
#[cached(result = true)]
```

```
fn expensive_2(n: usize) -> Result<usize, String> { ...  
}
```

Cache Custom Type - Error

```
enum Choice {  
    A,  
    B,  
    C,  
}
```

```
#[cached]
```

```
fn expensive(choice: Choice) -> usize { ...  
}
```

Error

```
error[E0277]: the trait bound `Choice: Hash` is not satisfied
--> src/bin/5.rs:13:1
   |
13 | #[cached]
   | ^^^^^^^^^ the trait `Hash` is not implemented for `Choice`
```

```
error[E0277]: the trait bound `Choice: Eq` is not satisfied
--> src/bin/5.rs:13:1
   |
13 | #[cached]
   | ^^^^^^^^^ the trait `Eq` is not implemented for `Choice`
```

Cache Custom Type - Fixed

```
#[derive(Clone, Eq, Hash, PartialEq)]  
enum Choice {  
    A,  
    B,  
    C,  
}
```

```
#[cached]  
fn expensive(choice: Choice) -> usize { ...  
}
```

Cannot Cache Borrowed Data

```
use cached::proc_macro::cached;
```

```
#[cached]
```

```
fn expensive(s: &str) -> &str {
```

```
    s
```

```
}
```

```
error[E0621]: explicit lifetime required in the type of `s`
```

```
--> src/bin/6.rs:6:1
```

```
6 | #[cached]
```

```
   ^^^^^^^^^^^ lifetime `'static` required
```

```
7 | fn expensive(s: &str) -> &str {
```

```
   |             ---- help: add explicit lifetime `'static`
```

Recap

- ◆ *cached* offers a way to automatically cache return values from functions
 - Can set cache capacity limits & TTL
- ◆ Uses a HashMap under the hood
 - Return values must be owned
 - Function parameters must be hashable
 - ▶ `#[derive(Clone, Eq, Hash, PartialEq)]`