Borrowing & Owning

CIS 198 Lecture 17

Borrowing vs. Owning

- Consider this struct:

```
struct StrToken<'a> {
    raw: &'a str,
}

impl<'a> StrToken<'a> {
    pub fn new(raw: &'a str) -> StrToken<'a> {
        StrToken { raw: raw, }
    }
}

// ...

let secret: String = load_secret("api.example.com");
let token = StrToken::new(&secret[..]);
```

*Code and examples taken from From &str to Cow

Borrowing vs. Owning

```
struct StringToken {
    raw: String,
}

impl StringToken {
    pub fn new(raw: String) -> StringToken {
        StringToken { raw: raw, }
    }
}
```

std::convert::Into & std::convert::From

```
pub trait Into<T> {
    fn into(self) -> T;
}

pub trait From<T> {
    fn from(T) -> Self;
}

impl<T, U> Into<U> for T where U: From<T> { /* ... */ }
```

Borrowing \/ Owning

```
struct Token {
    raw: String,
}

impl Token {
    pub fn new<S: Into<String>>(raw: S) -> Token
        Token { raw: raw.into(), }
    }
}
```

Bovine Intervention!

```
pub enum Cow<'a, B> where B: 'a + ToOwned + ?Sized {
    Borrowed(&'a B),
    Owned(B::Owned),
}
```

Bovine Intervention!

```
struct Token<'a> {
    raw: Cow<'a, str>,
}

impl<'a> Token<'a> {
    pub fn new<S: Into<Cow<'a, str>>(raw: S) -> Token<'a>
        Token { raw: raw.into(), }
    }
}

// ...

let token = Token::new(Cow::Borrowed("it's a secret"));
let secret: String = load_secret("api.example.com");
let token = Token::new(Cow::Owned(secret));
```

Recap

- If you want to have the potential to own *or* borrow a value in a struct, `Cow` is right for you.
- `Cow` can be used with more than just `&str`/`String`, though this is a very common usage.
- When you mutate a `Cow`, it might have to allocate and become its `Owned` variant.
- The type inside of the `Cow` must be convertible to some owned form.