Episode 09: Sigils

Sigils

Sigils are a way to create a shorthand.

```
~r/hello

# With a sigil, this becomes:
%Regex{opts: [], re_pattern: {re_pattern, 0, 0, ...}}
```

Anatomy of a sigil

```
~r/content/opts
```

1. tilde, ~
2. a letter
3. content wrapped by delimiters
4. options after the last delimiter

Eight delimiters are supported.

```
~r/content/i       ~r(content)i
~r|content|i       ~r[content]i
~r"content"i       ~r{content}i
~r'content'i       ~r<content>i
```

Interpolation

Lowercase sigils allow interpolation, but uppercase sigils do not.

```
word = "food"

~w(I love #{word})
# => ["I", "love", "food"]

~W(I love #{word})
# => ["I", "love", "\#{word}"]
```

Built-in sigils

Built-in sigils are defined in the `Kernel` module, which is automatically imported into every module.

```
defmodule MyModule do
  import Kernel # This happens implicitly on every module
end
```

- **~r for regular expressions.**
- **~w for lists of strings.**
  - Add the "a" option to make a list of atoms
  - `~w(hello there)a # => [:hello, :there]`
- **~s for strings which contain " symbols.**
  - Useful for documentation with a lot of quote symbols.
- **~c for character lists that include apostrophes.**

Writing sigils

`~r/hello/` calls your module's `sigil_r/2` function.

```
~r/hello/im

# Is transformed by the compiler to:
sigil_r("hello", 'im')

# Which returns:
%Regex{opts: [], re_pattern: {:re_pattern, 0, 0, ...}}


# Define them
defmodule MySigils do
  def sigil_u(...)
  def sigil_a(...)
end

# Import them
defmodule UserModule do
  import MySigils
  def some_function do
    ~u(hello there)
    ~a[content]
  end
end
```

Overriding built-in sigils

Don’t do this! Just know that it is possible.

```
defmodule MyModule do
  import Kernel, except: [sigil_r, 2]

  def sigil_r(content, opts) do
    "Hello World"
  end

  def use_sigil do
    ~r/hello/ # uses our function, not Kernel's
  end
end
```

> **Note:** There is no global way to do this. In order to prevent bugs and confusing code, Elixir locks down this functionality to a single module.