Pattern Matching

This is something that will be new to even experienced programmers.

It's also something that is extremely powerful and defines many constructs and a way of thinking about Elixir programs.

Let's see an example:

```
x = 25        # This is actually a match operator

25 = x        # Perfectly valid!

30 = x        # MatchError (Assignment only happens to the left of =)

x = 36        # Re-assign x to make the match true
```

Putting the Match Op to Use

Elixir typically uses pattern matching in place of exceptions and error (even though exceptions are there as well).

```
# Let's say you call a function to open a file
# It can return two types of values

a = {:ok, "File contents"}
b = {:error, "Error description"}

# You can check which one was returned and extract info from it
{:ok, x} = a
x

{:error, y} = b
y

# If :ok was returned, the :error tuple will not match (and vice versa)
```

Matching Lists

We can also extract head and tails instead of using `hd` and `tl`.

```
[head | tail] = [1, 2, 3]
head
tail

[head | _] = [1, 2, 3]
```

```
head
# Can't read from special variable: _
```

Supressing Assignment

```
x  = 35
^x = 25    # Only try to match existing value, don't rebind
```

Pattern Matching with Case

```
a = {:ok, "File contents"}

case a do
   {:ok, x} ->
      "Success. Contents: " <> x
   {:error, x} ->
      "Error. Message: #{x} "
   _ ->
      "Default value"
end

a = {:error, "Error description"}
case a do
   {:ok, x} ->
      "Success. Contents: " <> x
   {:error, x} ->
      "Error. Message: #{x} "
   _ ->
      "Default value"
end

a = {:info, "No response."}
case a do
   {:ok, x} ->
      "Success. Contents: " <> x
   {:error, x} ->
      "Error. Message: #{x} "
   _ ->
      "Default value"
end
```

(Also lookup: Guards: https://elixir-lang.org/getting-started/case-cond-and-if.html)

"Elixir and Phoenix: Real World Functional Programming"

Video Course by Dr. Nauman

`http://recluze.net/learn`