Pattern Matching

.* you want to know

Text matching: Globbing

Presentation-1.pdf

presentation.pdf

Presentation-1.pdf

presentation-1.pdf

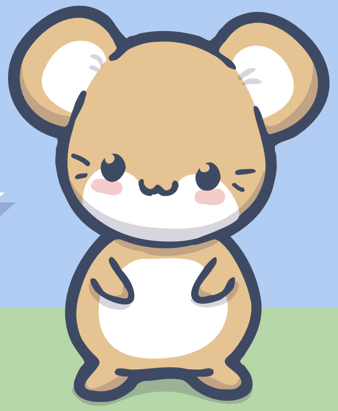
presentation-*.pdf

presentation.*

presentation-2.pdf

presentation-2.*

xyz



Text matching: Regular expressions

Presentation-1.pdf

presentation.pdf

Presentation-1.pdf

1

^P.*$

presentation-\d.pdf

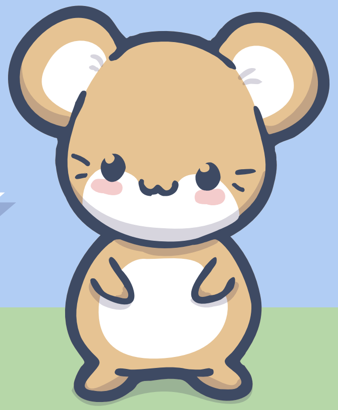
presentation.*

.*

presentation-2.pdf

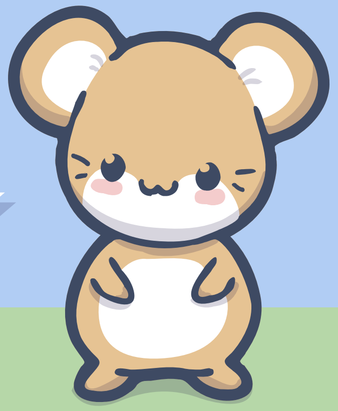
presentation-[^1]

xyz



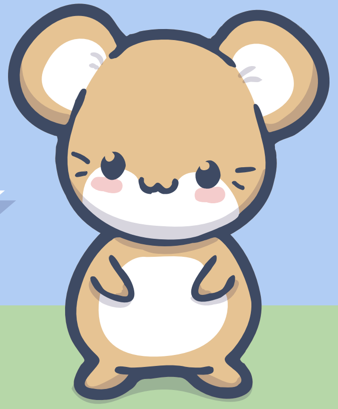
C-style languages

```
static const char * const band_description(enum band b)
{
  switch (b)
  {
    case low:     return "low";     break;
    case medium:  return "medium";  break;
    case high:    return "high";    break;
    default:      return "unknown"; break;
  }
}
```



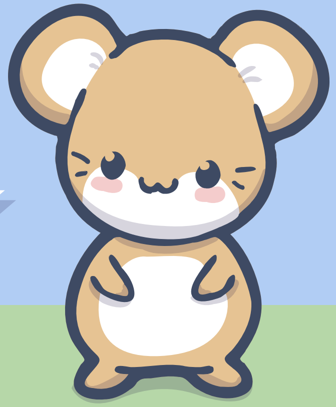
C-style languages

```
string Description(Band band, int n)
{
    switch (band)
    {
        case Band.Low:
            return "Low";

        case Band.Medium:
            return "Medium";

        case Band.High:
            return "High";

        default:
            return "wat";
    }
}
```



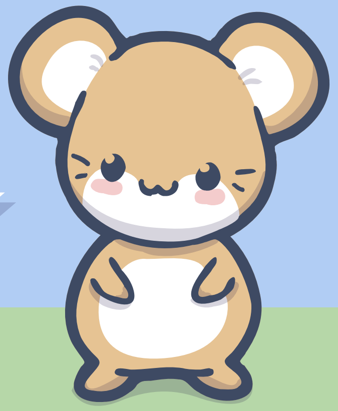
But that’s too static

```
enum band band(int n)
{
  switch (n)
  {
    case <10:   return low;     break;
    case <20:   return medium;  break;
    default:    return high;    break;
  }
}
```



cough

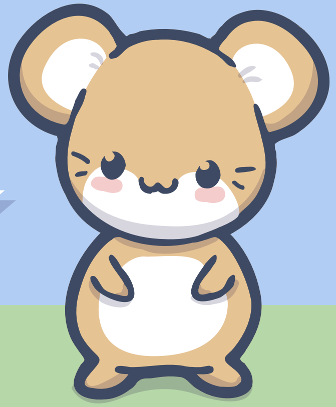
```
Function Band(b)
    Select Case b
        Case 0, 1, 2, 3
            Band = BandEnum.Low
        Case 4 To 9
            Band = BandEnum.Medium
        Case Is > 9
            Band = BandEnum.High
        Case Else
            Throw New FileNotFoundException
    End Select
End Function
```



Elixir

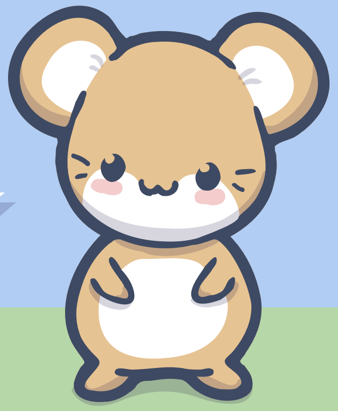
More highly-regarded languages have pattern matching, too

```
defmodule Band do
  def band(b) do
    cond do
      b in 0..9   -> :low
      b in 10..19 -> :medium
      true        -> :high
    end
  end
end
```



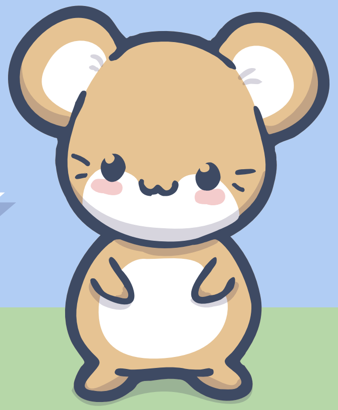
C#7

```
string Description(ILogger logger)
{
    switch (logger)
    {
        case FileLogger fileLogger:
            return fileLogger.Path;

        case SocketLogger socketLogger:
            return socketLogger.IP.ToString();

        default:
            return $"Unknown logger type: {logger.GetType().Name}";
    }
}
```



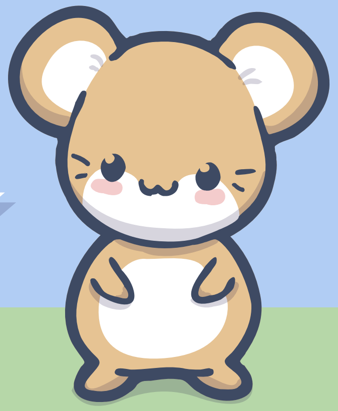
Guard clauses - beginnings

```
def description(band) do
  case band do
    :low    -> 'low'
    :medium -> 'medium'
    :high   -> 'high'
    _       -> 'wat'
  end
end
```



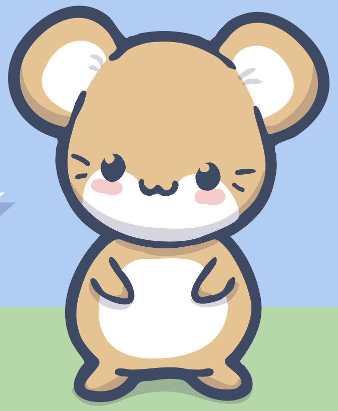
Guard clauses - Elixir

```
def description(band, n) do
  case band do
    :low    -> 'low'
    :medium -> 'medium'

    :high   when n > 100 -> 'really high'
    :high   -> 'high'

    _       -> 'wat'
  end
end
```



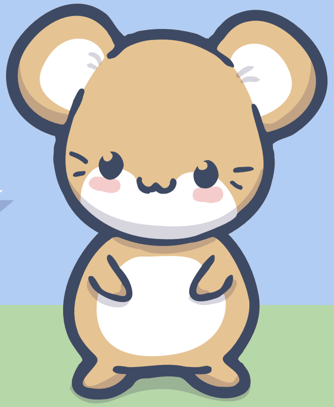
Guard clauses - Elixir

```
def description(band, n) do
  case band do
    :low    -> 'low'
    :medium -> 'medium'

    :high   when n > 100 -> 'really high'
    :high   -> 'high'

    _       -> 'wat'
  end
end
```



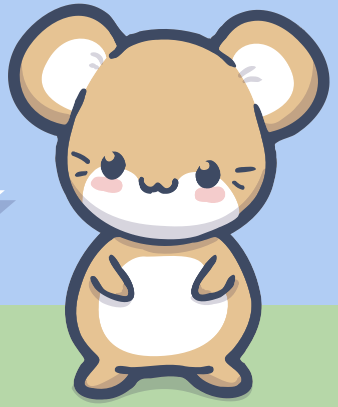
Guard clauses – C#

```
string Description(Band band, int n)
{
    switch (band)
    {
        case Band.Low:
            return "Low";

        case Band.Medium:
            return "Medium";

        case Band.High when n > 100:
            return "Really high";

        case Band.High:
            return "High";

        default:
            return "wat";
    }
}
```



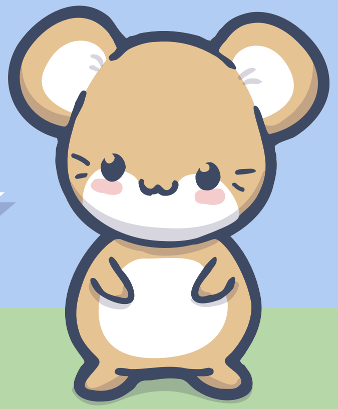
Matching in method signatures

```
def description(:low) do
  'low'
end

def description(:medium) do
  'medium'
end
```



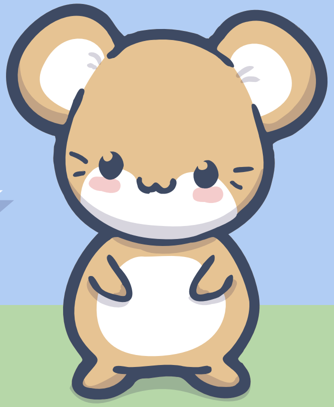
Matching in method signatures

```
def description(:low), do: 'low'
def description(:medium), do: 'medium'
```



MOAR

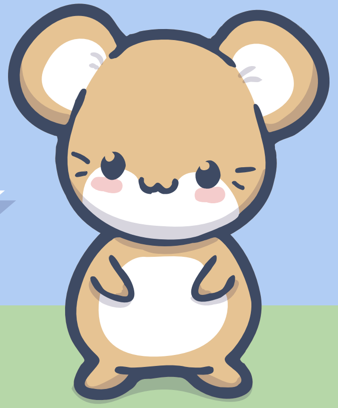
```elixir
def description(:low, n), do: 'low'
def description(:medium, n), do: 'medium'
def description(:high, n) when n > 100, do: 'really high'
def description(:high, n), do: 'high'
def description(_, n), do: 'wat'
```



But why?

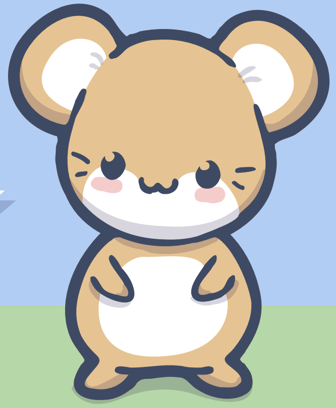
```
defmodule Maths do
  def divide(dividend, divisor) do
    dividend / divisor
  end
end

IO.puts Maths.divide 1, 2
IO.puts Maths.divide 1, 0
```

```
** (ArithmeticError) bad argument in arithmetic expression
```



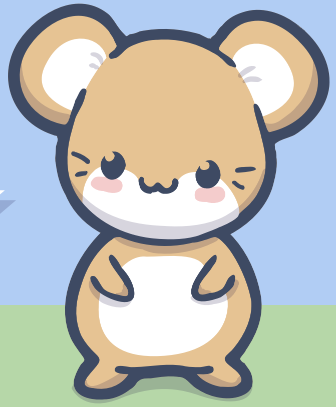
Because decluttering

```
defmodule Maths do
  def divide(_, 0) do
    0
  end
  def divide(dividend, divisor) do
    dividend / divisor
  end
end
```



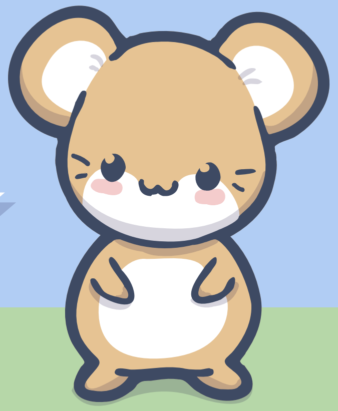
Idiomatic?

```
defmodule Log do
  def write(level, text) do
    IO.puts "#{level}: #{text}"
  end
end

Log.write :fail, "Human error"
```



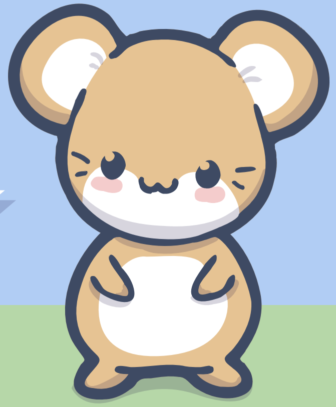
Just overloading based on signature

```
defmodule Log do
  def write(level, {code, text}) do
    IO.puts "#{level}: Error code #{code}: #{text}"
  end
  def write(level, text) do
    IO.puts "#{level}: #{text}"
  end
end
```



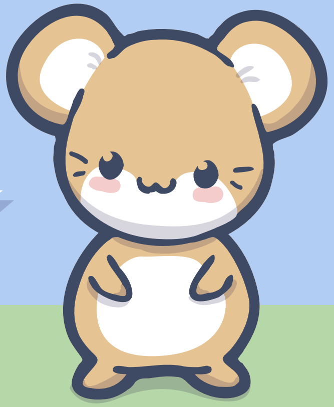
Hash, tuple or string

```
defmodule Log do
  def write(level, %{code: code, text: text}) do
    IO.puts "#{level}: Error code #{code}: #{text}"
  end
  def write(level, {code, text}) do
    IO.puts "#{level}: Error code #{code}: #{text}"
  end
  def write(level, text) do
    IO.puts "#{level}: #{text}"
  end
end
```



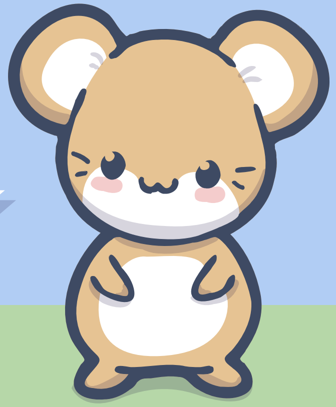
Match on value contained in value (!)

```
defmodule Log do
  def write(_, %{code: 42, text: text}) do
    raise FatalError(text)
  end
  def write(level, %{code: code, text: text}) do
    IO.puts "#{level}: Error code #{code}: #{text}"
  end
  def write(level, {code, text}) do
    IO.puts "#{level}: Error code #{code}: #{text}"
  end
  def write(level, text) do
    IO.puts "#{level}: #{text}"
  end
end
```



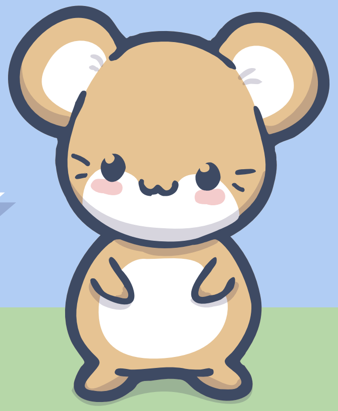
Destructuring return values

```
(bool ok, string description) Description(ILogger logger)
{
    switch (logger)
    {
        case FileLogger fileLogger:
            return (true, fileLogger.Path);

        case SocketLogger socketLogger:
            return (true, socketLogger.IP.ToString());

        default:
            return (false, logger.GetType().Name);
    }
}
```



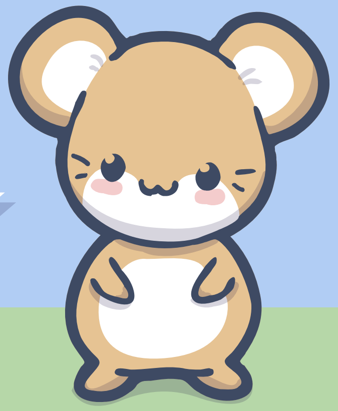
Destructuring return values

```
void Main()
{
    (var ok, var description) = Description(null);

    if (ok)
    {
        // Print it?
    }
    else
    {
        // Raise alarm?
    }
}
```



Destructuring return values

```
(result, description) = description(null)

case result do
  :ok -> ...
  _ -> ...
end
```



Matching return values

```
(:ok, description) = description(null)
```