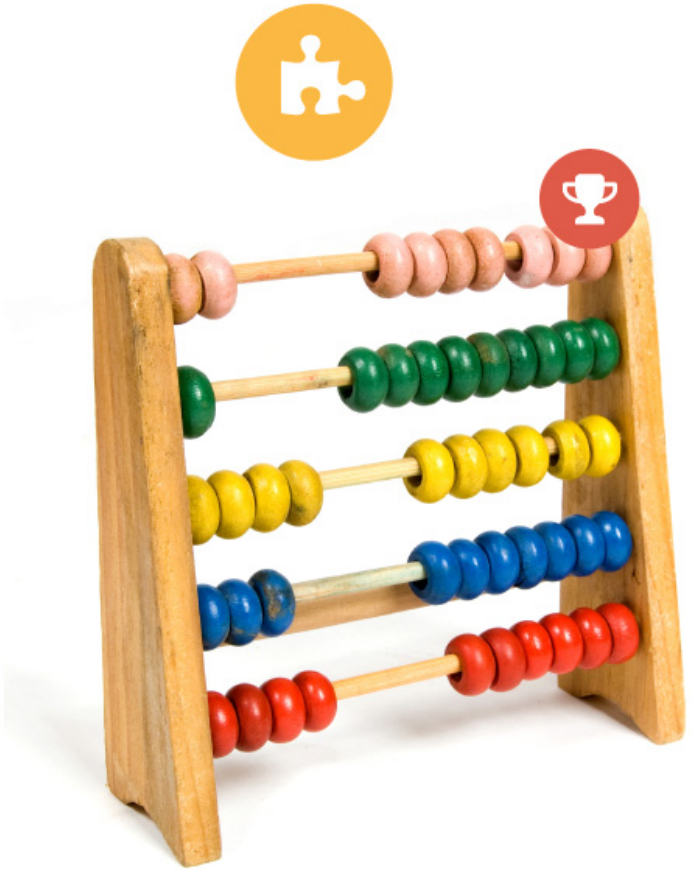
Programación en Lenguaje Ruby

Excepciones

Delio Tolivia

Gijón crece 2016 / 2019 AYTO. DE GIJÓN / CCOO / FADE / UGT
Ayuntamiento de Gijón
Gijón impulsa Empleo
dicampus TALENTO FORMACIÓN



Índice

Excepciones

begin, rescue end

ensure

raise

Excepciones propias

throw catch

Excepciones

- Las excepciones se lanzan cuando ocurre un error en el programa. Inmediatamente paran la ejecución del mismo

- Las excepciones se propagan por la pila de llamadas (de donde se generaron al método que la invoco, y así sucesivamente hasta el punto en que se inicio el programa (se puede ver con el método ***backtrace*** de la excepción)

begin, rescue end (I)

- Para parar una excepción podemos utilizar el bloque ***begin ... rescue … end***

```
def start_summer
  raise Exception.new("overheated!")
end

begin
  start_summer
rescue Exception => e
  puts "Let me tell you about heat! #{e.inspect}"
End


# Let me tell you about heat! #<Exception: overheated!>
```

begin, rescue end (II)

- Cuando queremos utilizar esto en un método las palabras reservadas begin y end no se necesitan pues se da por supuesto que se quieren capturar las excepciones en todo el método

```
def decode_all(secrets)
  secrets.map {|s| decode(s) }
rescue
  puts "whew! safe."
end
```

ensure

- Cuando hay un código que queremos que se ejecute tanto cuando ocurre una excepción como cuando no (por ejemplo cerrar una conexión a una base de datos, grabar un fichero, etc…) tenemos la palabra reservada ensure

```
class UserDataAccess
  attr_accessor :db

  def initialize
    @db = Database.new
  end

  def find_user(name)
        @db.find("SELECT * FROM USERS WHERE NAME = '%'", name)
  rescue Exception => e
    puts "A database error occurred."
  ensure
        @db.close
  end
end
```

raise

- Podemos lanzar nuestras excepciones con ***raise***, ***Kernel.raise*** o ***Kernel.fail***.
- ***raise*** lanza la excepción ***RunTimeError***, con el mensaje que le pasemos como argumento

```
raise "Rise."
RuntimeError: Rise
```

- También podemos indicar cual es la excepción que lanza

```
raise StandardError, "Raising standard error”
StandardError: Raising standard error
```

Excepciones propias

- La clase ***Exception*** tiene varias clases hijas y a su vez estas otras más. Para crear un tipo de excepción propia podemos heredar de ***Exception*** o de alguna de estas como ***StandardError***

```
class SomeCustomError < StandardError
end
```

- Es buena costumbre acabar el nombre con la palabra Error

throw / catch (I)

- Puede pensarse en utilizar las excepciones dadas hasta ahora para romper la ejecución de un bucle cuando queramos. Eso no es correcto. Si queremos usarlo tenemos las instrucciones throw/ catch.

```
floor = [["blank", "blank", "blank"],
        ["gummy", "blank", "blank"],
        ["blank", "blank", "blank"]]

attempts = 0
candy = nil
catch(:found) do #Indicamos el símbolo que debe lanzar el throw para capturarlo
        floor.each do |row|
          row.each do |tile| #Con un break solamente saldriamos de este bucle
            attempts += 1
            if tile == "jawbreaker" || tile == "gummy"
              candy = tile
              throw(:found) #Cuando la encontramos no hacemos el resto de iteeraciones
            end
          end
        end
end
puts candy
puts attempts
```

throw / catch (II)

- Otra cosa que podemos hacer con un throw es devolver un valor que capturar el catch y lo devolverá. Sería como utilizarlo a modo de return.

```
floor = [["blank", "blank", "blank"],
         ["gummy", "blank", "blank"],
         ["blank", "blank", "blank"]]

attempts = 0
candy = catch(:found) do
          floor.each do |row|
          row.each do |tile|
        attempts += 1
          throw(:found, tile) if tile == "jawbreaker" || tile == "gummy"
          end
          end
end
puts candy
puts attempts
```