Dibujando con ciclos anidados

Objetivos

- Reconocer patrones que requieran de más de un ciclo para resolverlos
- Utilizar ciclos anidados para dibujar patrones

Introducción

Los desafíos de dibujo son ideales para practicar ciclos y especialmente ciclos anidados. ¿Qué tipo de cosas podemos dibujar con ciclos anidados?

Veamos el siguiente patrón:

```
*****
*****
*****
*****
*****
```

Solución

```
n = 5
# n = ARGV[0]

n.times do |i|
    n.times do |j|
        print '*'
    end
    print "\n"
end
```

Desafío medio triángulo

Crear el programa `medio_triangulo.rb` que reciba el tamaño del triángulo y dibuje el siguiente patrón:

Ejemplo:

`ruby medio_triangulo.rb 5`

```
*
**
***
****
*****
```

Soludicón desafío medio triángulo

Para resolver el código tenemos que observar el patrón:

- Si el usuario ingresa 5, se dibujan 5 filas.
- En la fila 1, hay un `*`
- En la fila 2, hay 2 `*`
- Por lo que podemos decir que en la fila n y hay n `*`

Las n filas

Primer acercamiento a la solución con n filas

```
 n = ARGV[0] # 6

n.times do |i|
    puts '*'
end
```

```
*
*
*
*
*
*
```

Las columnas

```
 n = ARGV[0]

n.times do |i|
    # Cuando i es 1 repetimos 1 vez
    # Cuando i es 2 repetimos 2 veces
    # Cuando i es N repetimos N veces
    # O sea que siempre estamos repetiendo i veces
    i.times do |j|
      print '*'
    end
    print "\n"
end
```

```
*
**
***
****
*****
```

Desafío triángulo

```
*
**
***
****
*****
****
***
**
*
```

Solución desafío triángulo

Ya sabemos cómo hacer la primera mitad

Nos concentraremos en la segunda parte de triángulo, donde el patrón que tenemos que armar es:

```
*****
****
***
**
*
```

Si n es igual a 5 dibujaremos 5 asteriscos, luego 4, luego 3, o iremos decrementando de uno en uno. Esto es lo mismo que decir `n - i`

- Iteración 0:
  - 5 - 0 = 5
- Iteración 1:
  - 5 - 1 = 4
- Iteración 2:
  - 5 - 2 = 3
- Iteración n:
  - 5 - n

```
 n = ARGV[0] # 5

n.times do |i|
    (n-i).times do |j|
      print '*'
    end
    print "\n"
end
```

```
*****
****
***
**
*
```

Para armar el triángulo completo solo necesitamos juntar ambas soluciones:

```
 n = ARGV[0] # 6

n.times do |i|
    i.times do |j|
      print '*'
```

```
    end
    print "\n"
end

n.times do |i|
    (n - i).times do |j|
      print '*'
    end
    print "\n"
end
```

```
*
**
***
****
*****
******
*****
****
***
**
*
```

Desafío del cuadrado vacío

Crear el programa `cuadrado_hueco.rb` que al ejecutarse reciba un tamaño y dibuje un cuadrado dejando vacío el interior.

uso:

`cuadrado_hueco.rb 3`

```
resultado:
***
* *
***
```

`cuadrado_hueco.rb 5`

```
resultado:
*****
*   *
*   *
*   *
*****
```

pista: Identifica cuál es la parte que se repite.

Solución desafío cuadrado vacío

La parte superior e inferior son similares y siempre son dos. La parte del medio, sin embargo, contempla el resto del cuadrado.

Primero dibujaremos la parte superior e inferior.

```
n = 5
#n = ARGV[0]

# Parte superior
n.times do |i|
    print "*"
end
print "\n"

# Parte inferior
n.times do |i|
    print "*"
end
```

La parte del medio consta siempre de dos asteriscos, uno al principio y el otro al final. El resto son espacios en blanco.

```
- Si n es 4, hay 2 asteriscos y 2 espacios.
- Si n es 5, hay 2 asteriscos y 3 espacios.
- Si n es 6, hay 2 asteriscos y 4 espacios.
- Si n es 7, hay 2 asteriscos y 5 espacios.
```

La cantidad de espacios siempre es `n - 2`

```
# n = 5

print "*"
(n - 2).times do |i|
  print " "
end
print "*"
```

Luego tenemos que repetir lo mismo todo n - 2 veces.

```ruby
# n = 5
(n - 2).times do
  print "*"
  (n - 2).times do |i|
    print " "
  end
  print "*"
  print "\n"
end
```

```
*   *
*   *
*   *
```

Finalmente unimos todo el código agregando las tapas.

```ruby
n = ARGV[0].to_i

# Parte superior
n.times do
    print "*"
end
print "\n"

# Parte del medio
(n - 2).times do
  print "*"
  (n - 2).times do
    print " "
  end
  print "*"
  print "\n"
end

# Parte inferior
n.times do
    print "*"
end
```

Desafío listas

Se pide crear el programa `listas_y_sublistas.rb` donde el usuario ingrese un número como argumento y se genere una lista de HTML con esa cantidad de ítems y un segundo número que indique la cantidad de sub ítems.

Uso:

`ruby listas_y_sublistas.rb 3 2`

```
<ul>
    <li>
        <ul>
            <li> 1.1 </li>
            <li> 1.2 </li>
        </ul>
    </li>
    <li>
        <ul>
            <li> 2.1 </li>
            <li> 2.2 </li>
        </ul>
    </li>
    <li>
        <ul>
            <li> 3.1 </li>
            <li> 3.2 </li>
        </ul>
    </li>
</ul>
```

Pistas:

- Puedes tabular con `"\t"`
- Puedes hacer un salto de línea con `"\n"`

Solución desafío listas

Lo primero que tenemos que hacer es identificar las partes que se repiten de las que no. Dentro de cada lista hay un patrón que se repite

```
                <li> 1.1 </li>
                <li> 1.2 </li>
```

Pero además, fuera de la lista, hay otro patrón que se repite:

```
        <li>
            <ul>
                <li> 1.1 </li>
                <li> 1.2 </li>
            </ul>
        </li>
```

Así que ocuparemos 2 iteraciones: Una para el ciclo interior y otra para el ciclo exterior.

Hacemos la lista interna primero:

```
n_externo = ARGV[0]
n_interno = ARGV[1]

n_interno.times do |i|
  puts "<li> #{i} </li>"
end
```

El siguiente paso es agregar el ciclo externo.

```
n_externo = ARGV[0]
n_interno = ARGV[1]

n_externo.times do |j|
  puts "<li>\n"
  puts "\t<ul>"
  n_interno.times do |i|
    puts "\t\t<li> #{j}.#{i} </li>"
  end
  puts "\t</ul>"
  puts "</li>"
end
```

Finalmente agregamos las etiquetas `<ul> </ul>` al principio y al final respectivamente.

```ruby
n_externo = ARGV[0]
n_interno = ARGV[1]

puts "<ul>"
n_externo.times do |j|
  puts "<li>\n"
  puts "\t<ul>"
  n_interno.times do |i|
    puts "\t\t<li> #{j}.#{i} </li>"
  end
  puts "\t</ul>"
  puts "</li>"
end
puts "</ul>"
```