Ruby Loops

While Loop

A while loop is similar to the loop loop except that you declare the condition that will break out of the loop up front.

```
i = 0
while i < 10 do
 puts "i is #{i}"
 i += 1
end
```

You can also use while loops to repeatedly ask a question of the user until they give the desired response:

```
while gets.chomp != "yes" do
  puts "Will you go to prom with me?"
end
```

======

Until Loop

The until loop is the opposite of the while loop.
A while loop continues for as long as the condition is true,
whereas an until loop continues for as long as the condition is false.
These two loops can therefore be used pretty much interchangeably.
Ultimately, what your break condition is will determine which one is more readable.

As much as possible, you should avoid negating your logical expressions using ! (not).
First, it can be difficult to actually notice the exclamation point in your code.
Second, using negation makes the logic more difficult to reason through and therefore makes your code more difficult to understand. These situations are where until shines.

We can re-write our while loop examples using until.

```
i = 0
until i > 10 do
 puts "i is #{i}"
 i += 1
end
```

The next example shows how you can use until to avoid the negation ! that the above while loop had to use.

```
until gets.chomp == "yes" do
  puts "Will you go to prom with me?"
end
```

==========

Ruby Loops

Ranges

What if we know exactly how many times we want our loop to run?
Ruby lets us use something called a range to define an interval.
All we need to do is give Ruby the starting value, the ending value,
and whether we want the range to be inclusive or exclusive.

```
(1..5)      # inclusive range: 1, 2, 3, 4, 5
(1...5)     # exclusive range: 1, 2, 3, 4
```

We can make ranges of letters, too!

```
('a'..'d')  # a, b, c, d
```

=======

For Loop

A for loop is used to iterate through a collection of information such as an
array or range. These loops are useful if you need to do something a given number
of times while also using an iterator.

```
for i in 0..5
  puts "#{i} zombies incoming!"
end
```

=========

Times Loop

If you need to run a loop for a specified number of times,
then look no further than the trusty #times loop.
It works by iterating through a loop a specified number of times
and even throws in the bonus of accessing the number it’s currently iterating through.

```
5.times do
  puts "Hello, world!"
end

5.times do |number|
  puts "Alternative fact number #{number}"
end
```

Remember, loops will start counting from a zero index unless specified otherwise,
so the first loop iteration will output Alternative fact number 0.

============

Ruby Loops

Upto and Downto Loops

The Ruby methods #upto and #downto do exactly what you'd think they do from their names.
You can use these methods to iterate from a starting number either up to or
down to another number, respectively.

```
5.upto(10) {|num| print "#{num} " }       #=> 5 6 7 8 9 10

10.downto(5) {|num| print "#{num} " }     #=> 10 9 8 7 6 5
```

If you need to step through a series of numbers (or even letters) within a specific range,
then these are the loops for you.

============

LOOP

The loop loop (say what????) is Ruby's loop that just won't quit.
It's an infinite loop that will keep going unless you specifically
request for it to stop, using the break command. Most commonly,
break is used with a condition, as illustrated in the example below.

```
i = 0
loop do
  puts "i is #{i}"
  i += 1
  break if i == 10
end
```

You won't see this loop used much in Ruby. **If you find yourself using loop,
know that there is probably a better loop for you out there**,
like one of the more specific loops below.