Recursion

Devpoint Labs - Dave Jungst / Jake Sorce

What is Recursion

Recursion is used to invoke a function call on itself.

The idea is that the solution to a problem is the solution to smaller versions of the same problem.

Recursion must always have a base case. This will cause the recursive loop to exit without calling itself again.

Recursion is a divide and conquer approach to a problem.

Most recursive algorithms can be done iteratively but more often than not the recursive solution is more elegant.

Recursive countdown

```
def countdown(n)

 if n == 0

   0

 else

    puts n

   countdown(n -1)

end
```

The base case is 0 this means that once the method call is broken down to the smallest number then it will no longer recursively call itself.

When this method is called it will call itself over and over breaking the problem down smaller until it hits 0

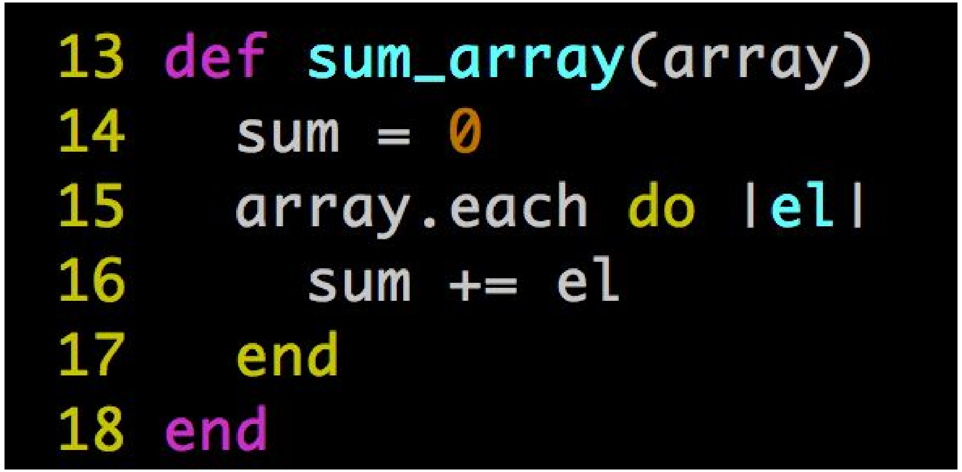
```
countdown(10)

countdown(9)

countdown(8)
```

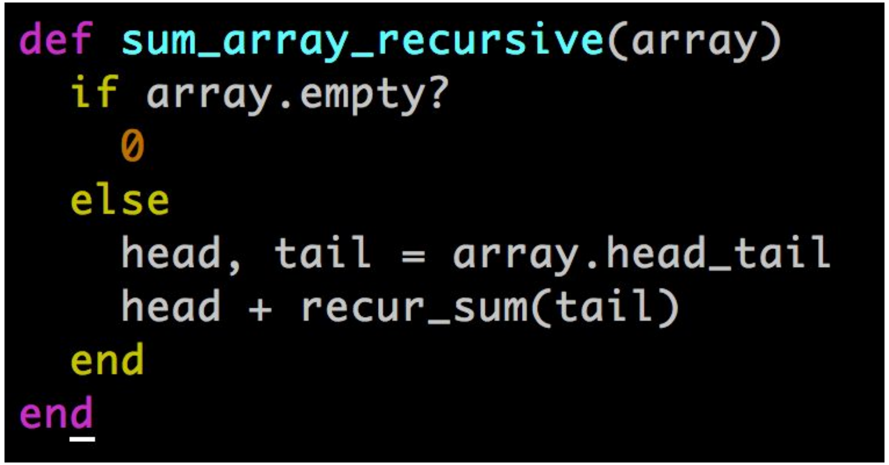
Summing an array

Iteratively



```ruby
def sum_array(array)
  sum = 0
  array.each do |el|
    sum += el
  end
end
```

Recursively



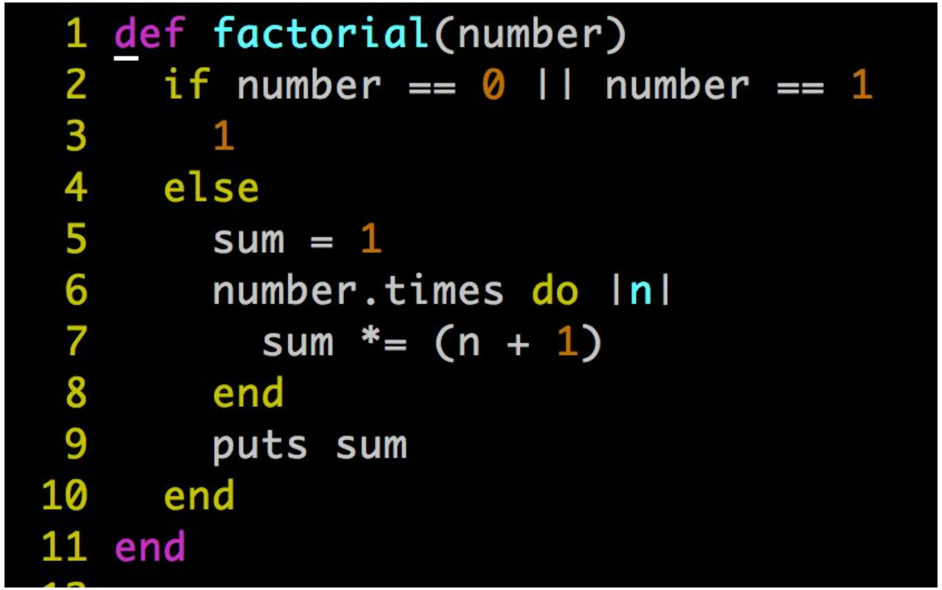
```ruby
def sum_array_recursive(array)
  if array.empty?
    0
  else
    head, tail = array.head_tail
    head + recur_sum(tail)
  end
end
```

Iterative Factorial

Factorial is the product of an integer and all integers below it.

For example 4! would be 4*3*2*1 = 24

Iterative factorial



```ruby
def factorial(number)
  if number == 0 || number == 1
    1
  else
    sum = 1
    number.times do |n|
      sum *= (n + 1)
    end
    puts sum
  end
end
```

Fibonacci Sequence

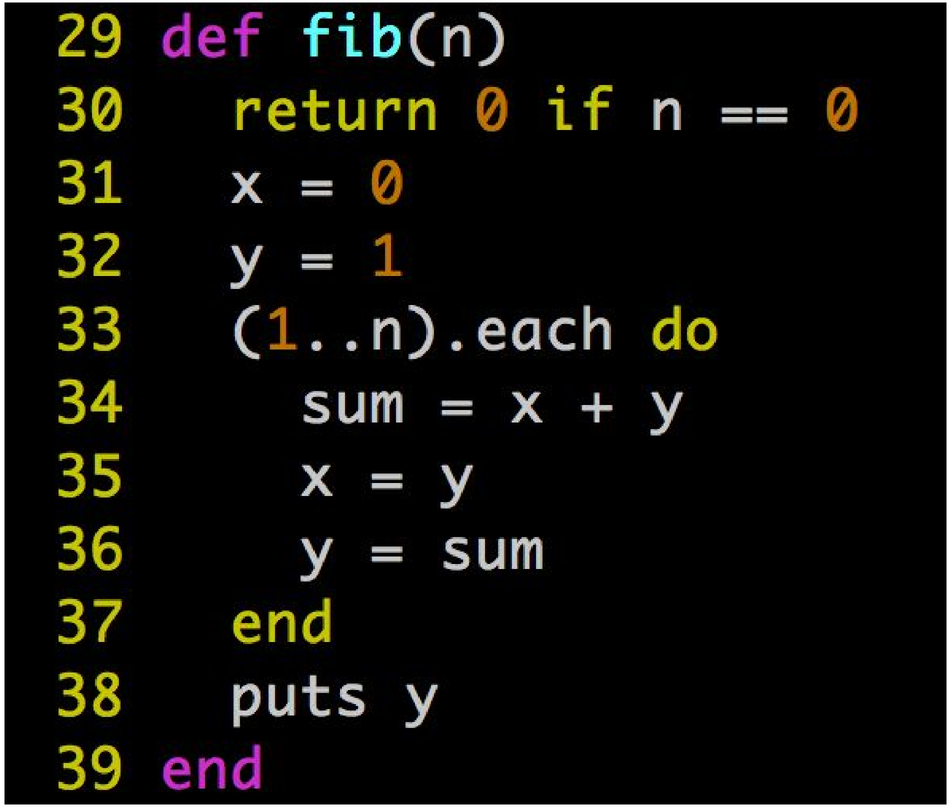
The Fib sequence… 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144…..

Is created by starting with 1 and then adding the 2 previous numbers to make the next number in the sequence

1, (0+1), (1 + 1), (2 + 1), (3 + 2), …….

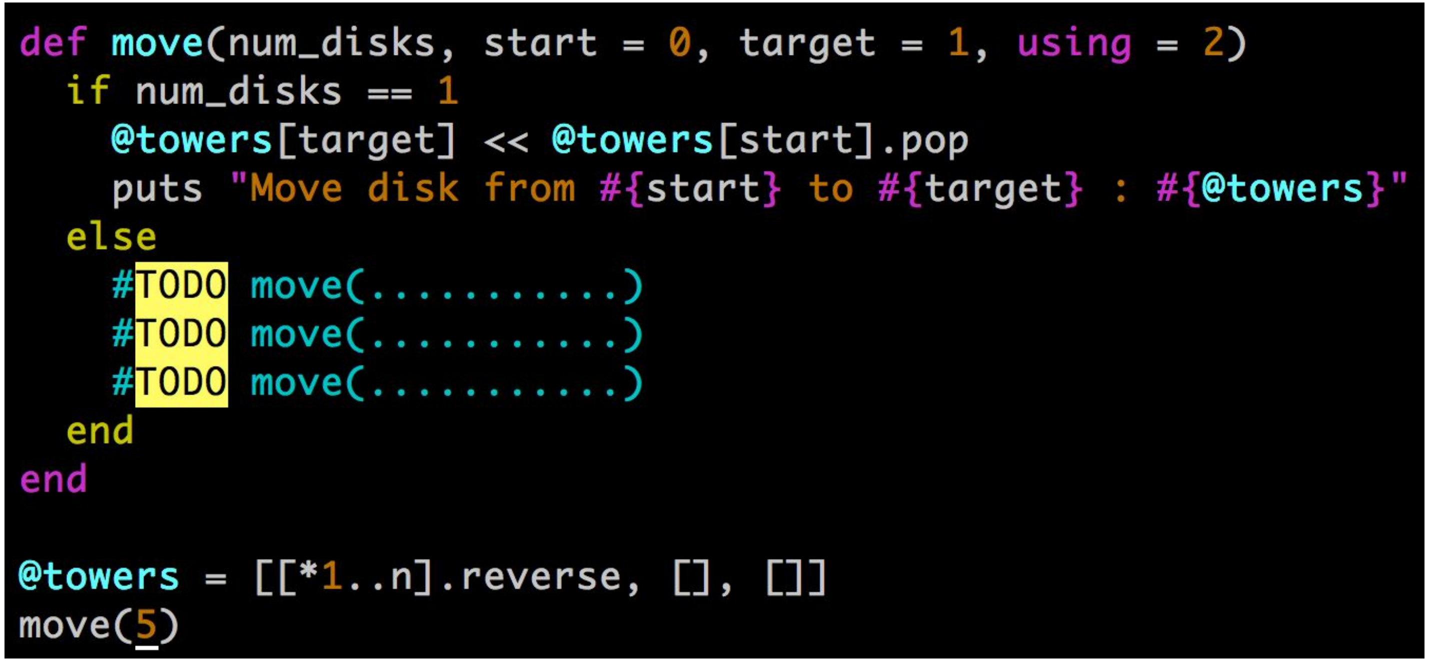
This problem can be solved both iteratively and recursively. The notion that the the original problem needs to be repeated and broken down until it reaches the base case suggests a good candidate for recursion

Fib iteratively



```ruby
def fib(n)
  return 0 if n == 0
  x = 0
  y = 1
  (1..n).each do
    sum = x + y
    x = y
    y = sum
  end
  puts y
end
```

Towers of Hanoi Recursive



```ruby
def move(num_disks, start = 0, target = 1, using = 2)
  if num_disks == 1
    @towers[target] << @towers[start].pop
    puts "Move disk from #{start} to #{target} : #{@towers}"
  else
    #TODO move(...........)
    #TODO move(...........)
    #TODO move(...........)
  end
end

@towers = [[*1..n].reverse, [], []]
move(5)
```

Code Challenge

Easy: Factorial Recursively

Medium: Fib Recursively

Hard: Towers of Hanoi