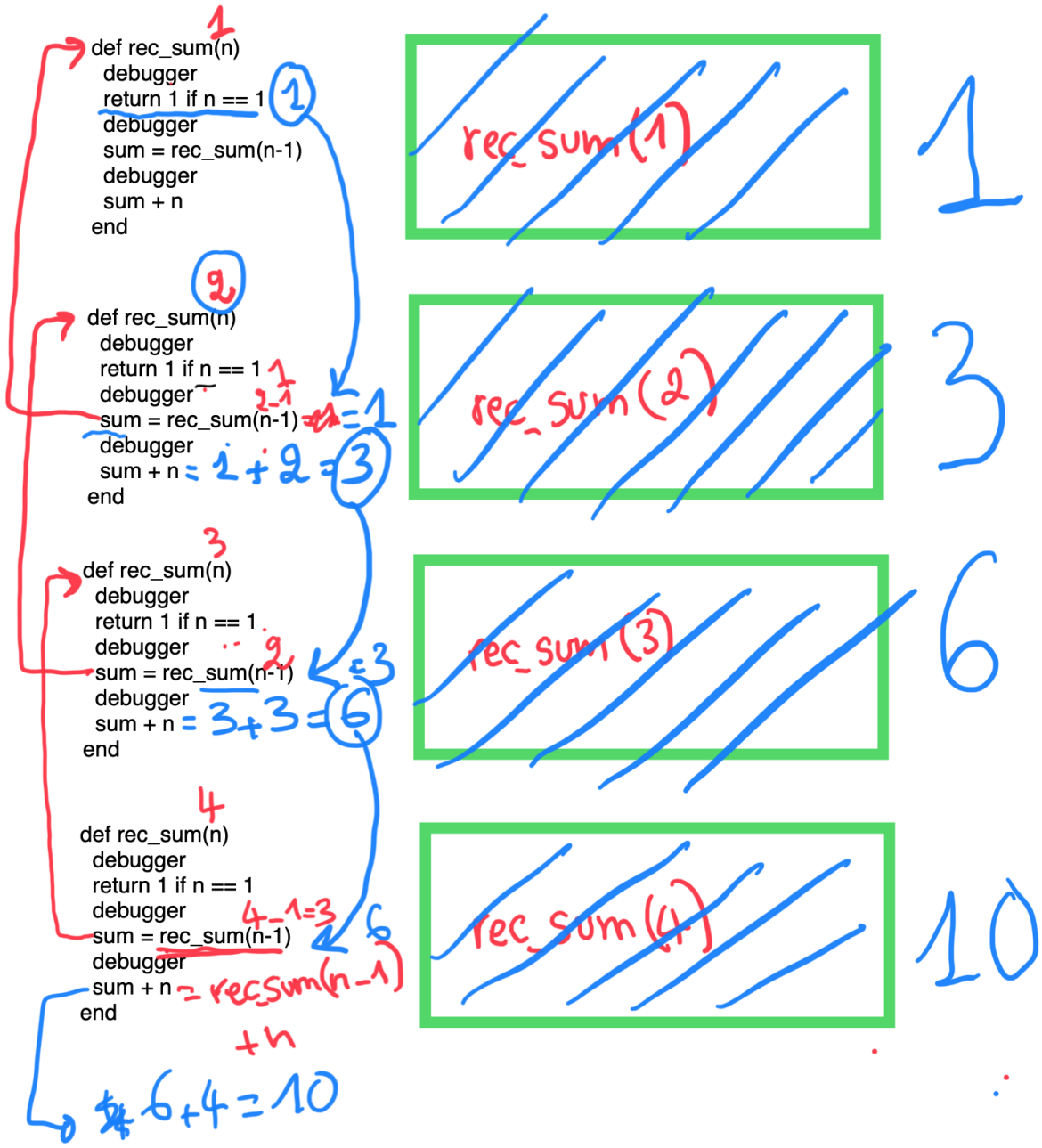
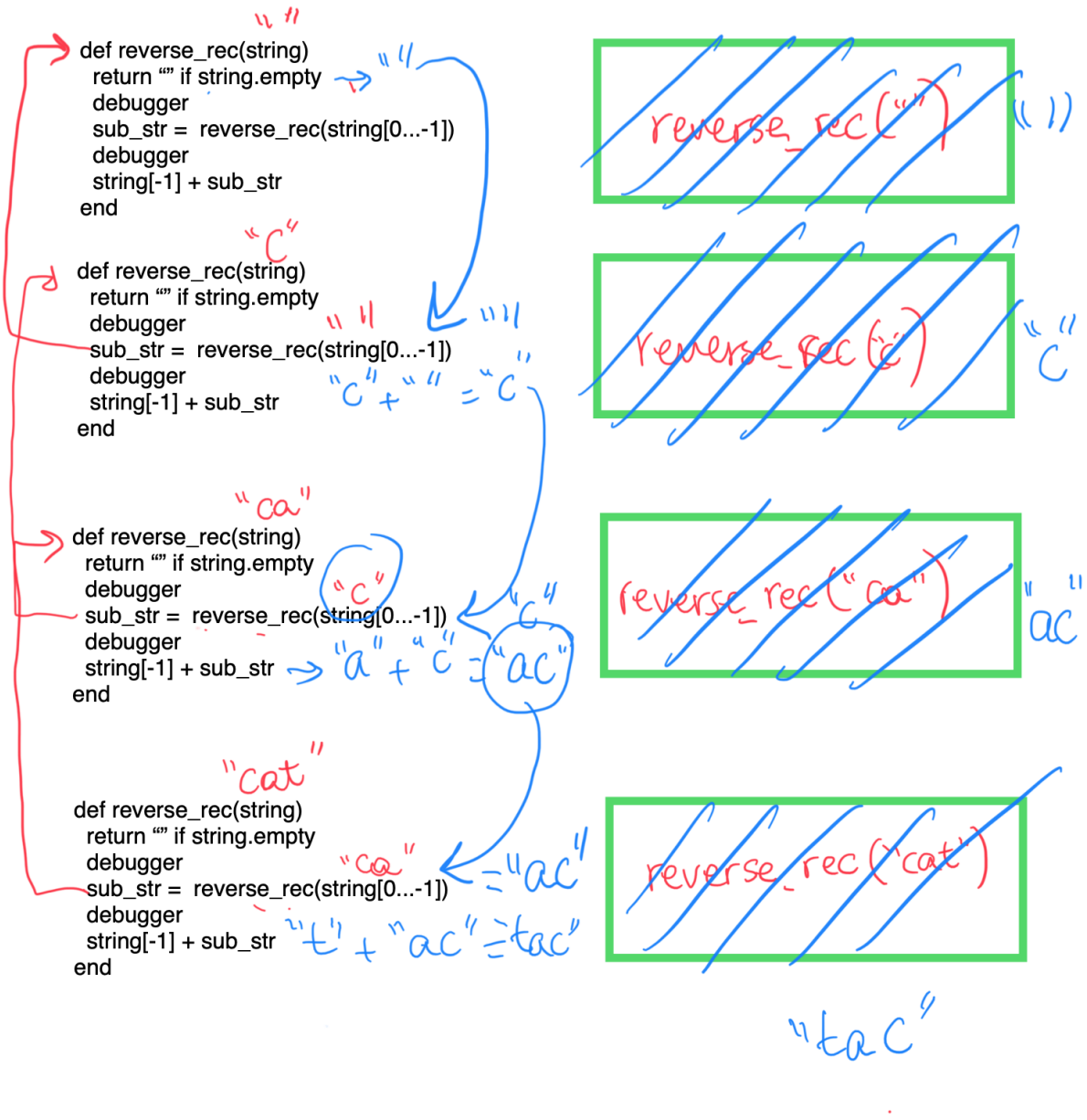
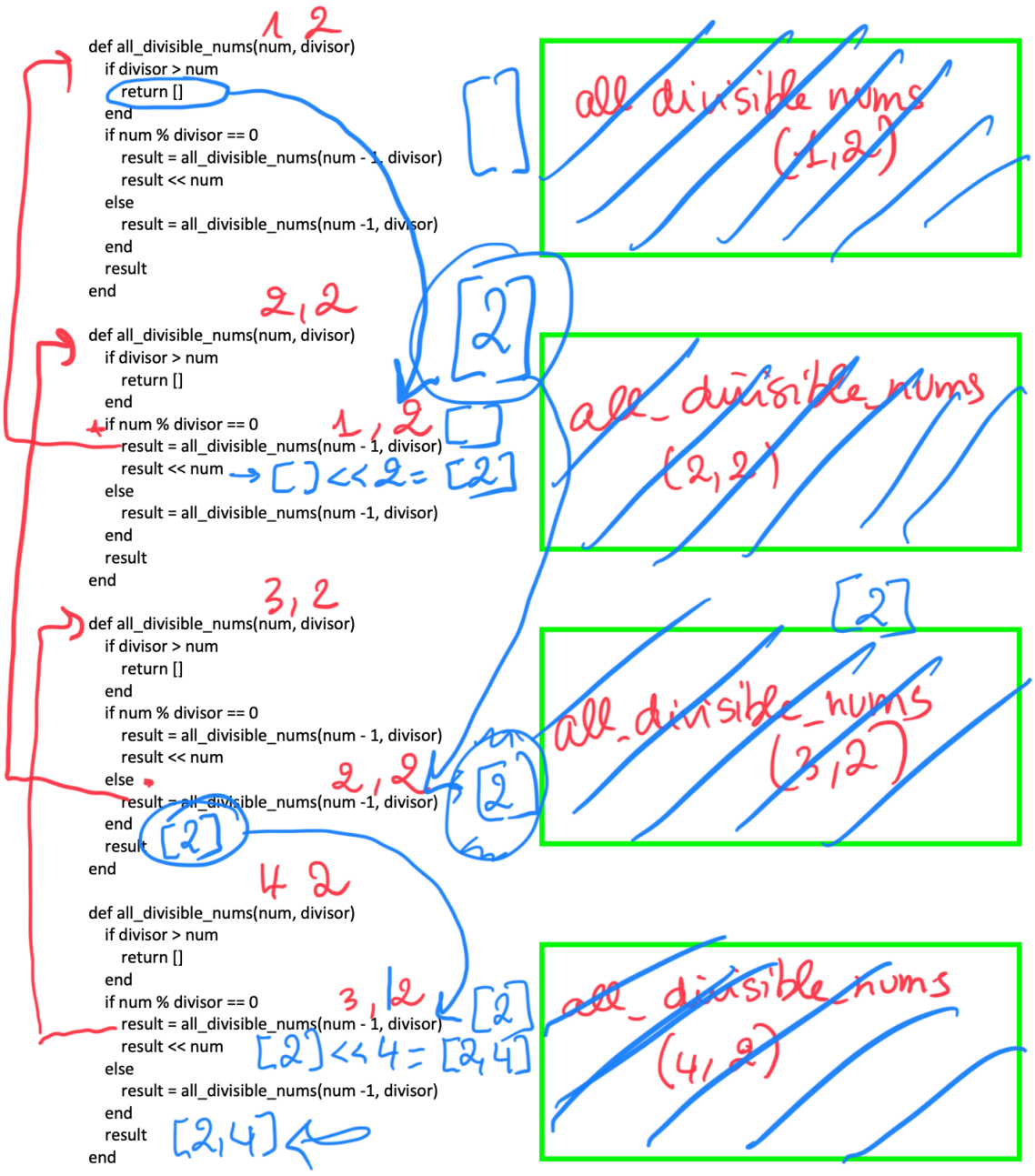
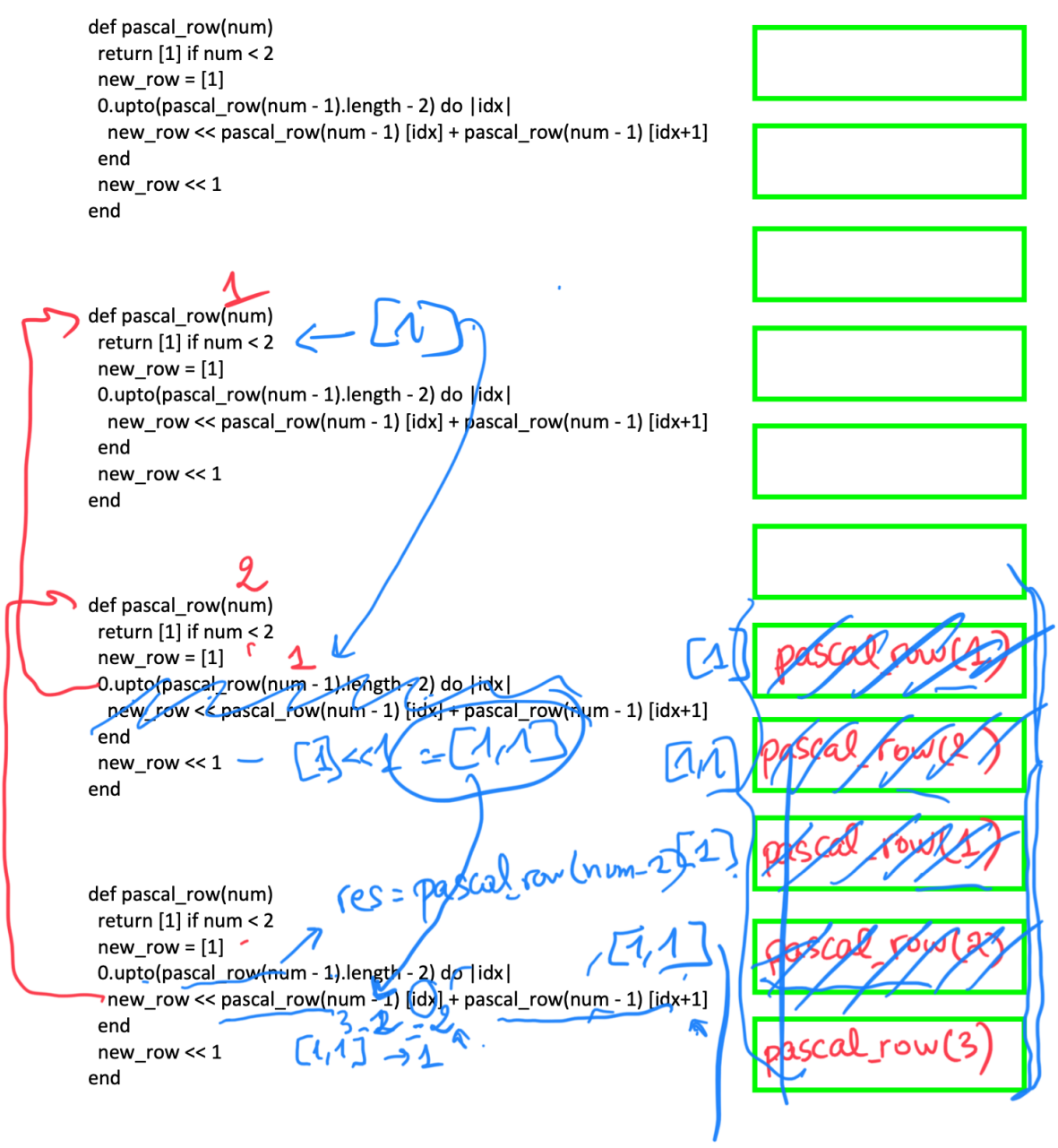
Recursion

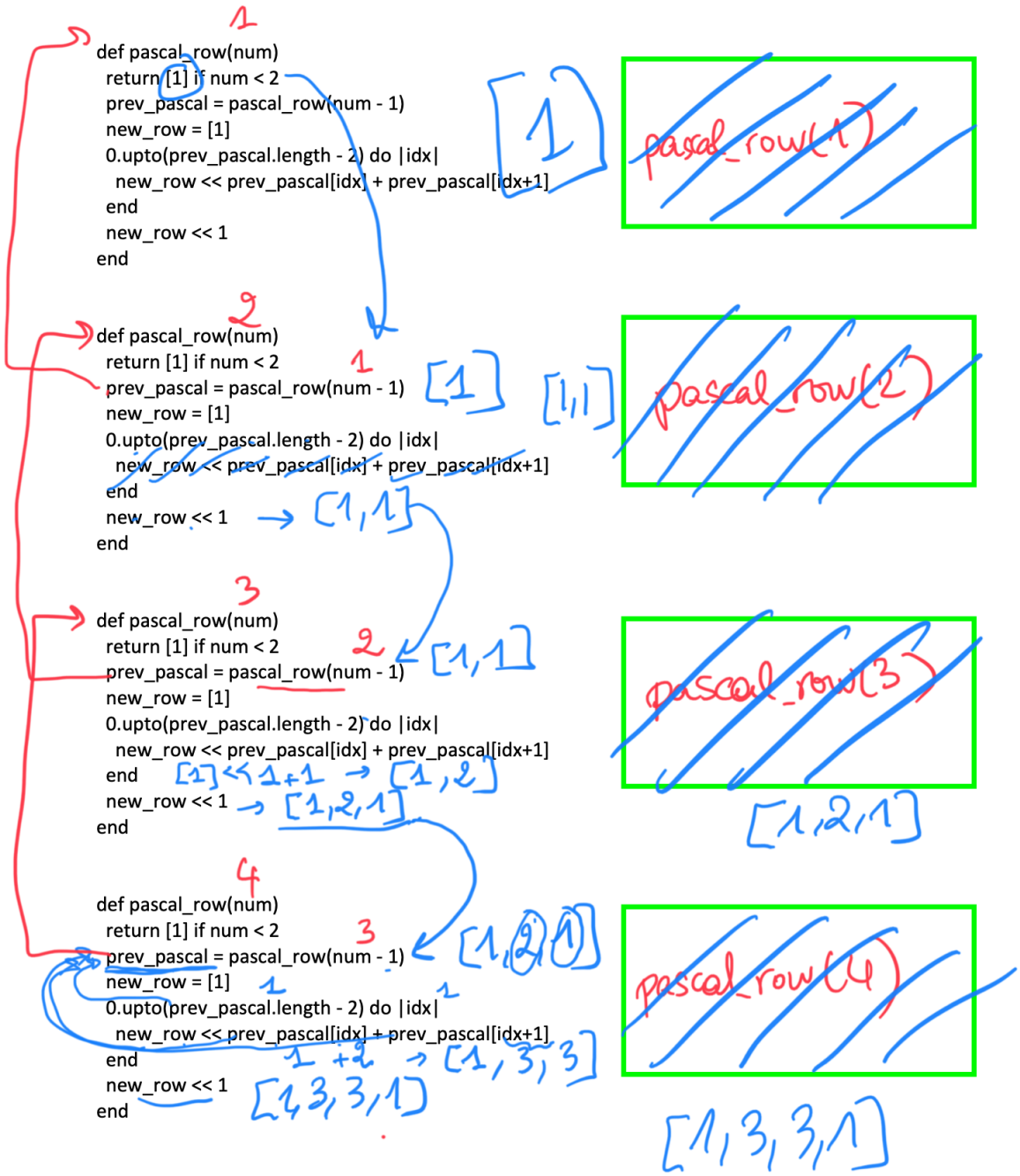

def rec_sum(n)
debugger
return 1 if n == 1
debugger
sum = rec_sum(n-1)
debugger
sum + n
end
rec_sum(1)
rec_sum(2)
rec_sum(3)
rec_sum(4)
1
3
6
10


def reverse_rec(string)
return "" if string.empty
debugger
sub_str = reverse_rec(string[0...-1])
debugger
string[-1] + sub_str
end
def reverse_rec(string)
return "" if string.empty
debugger
sub_str = reverse_rec(string[0...-1])
debugger
string[-1] + sub_str
end
def reverse_rec(string)
return "" if string.empty
debugger
sub_str = reverse_rec(string[0...-1])
debugger
string[-1] + sub_str
end
def reverse_rec(string)
return "" if string.empty
debugger
sub_str = reverse_rec(string[0...-1])
debugger
string[-1] + sub_str
end
reverse_rec("")
reverse_rec("c")
reverse_rec("ca")
reverse_rec("cat")
"c" + "" = "c"
"a" + "c" = "ac"
"t" + "ac" = "tac"
"tac"


1 2
def all_divisible_nums(num, divisor)
if divisor > num
return []
end
if num % divisor == 0
result = all_divisible_nums(num - 1, divisor)
result << num
else
result = all_divisible_nums(num -1, divisor)
end
result
end
2,2
def all_divisible_nums(num, divisor)
if divisor > num
return []
end
if num % divisor == 0
result = all_divisible_nums(num - 1, divisor)
result << num
else
result = all_divisible_nums(num -1, divisor)
end
result
end
1,2
[]<<2=[2]
3,2
def all_divisible_nums(num, divisor)
if divisor > num
return []
end
if num % divisor == 0
result = all_divisible_nums(num - 1, divisor)
result << num
else
result = all_divisible_nums(num -1, divisor)
end
result
end
2,2
[2]
4 2
def all_divisible_nums(num, divisor)
if divisor > num
return []
end
if num % divisor == 0
result = all_divisible_nums(num - 1, divisor)
result << num
else
result = all_divisible_nums(num -1, divisor)
end
result
end
3,2
[2]
[2]<<4=[2,4]
[2,4]
all_divisible_nums
(1,2)
[]
all_divisible_nums
(2,2)
[2]
all_divisible_nums
(3,2)
[2]
all_divisible_nums
(4,2)

```
def pascal_row(num)
 return [1] if num < 2
 new_row = [1]
 0.upto(pascal_row(num - 1).length - 2) do |idx|
  new_row << pascal_row(num - 1) [idx] + pascal_row(num - 1) [idx+1]
 end
 new_row << 1
end
```




def pascal_row(num)
return [1] if num < 2
prev_pascal = pascal_row(num - 1)
new_row = [1]
0.upto(prev_pascal.length - 2) do |idx|
new_row << prev_pascal[idx] + prev_pascal[idx+1]
end
new_row << 1
end
pascal_row(1)
pascal_row(2)
pascal_row(3)
pascal_row(4)
[1]
[1,1]
[1,2,1]
[1,3,3,1]