Programming for Data Science: Recursive function

Instructor: Jakramate Bootkrajang

Motivating example

- There are n people in a room. If each person shakes hands once with every other person. What is the total number h(n) of handshakes?

Problem breakdown

- If you can calculate the number of time that n-1 people shake hands, I can give you the answer
- Which is h(n-1) + n-1
  - In other word, h(n) = h(n-1) + n-1
- Now how to calculate h(n-1) ?

It’s the same problem

- Well, okay if you can calculate the number of times n-2 people shake hand, I can give you the answer
- which is h(n-2) + n-2

And the list goes on

- h(n) = h(n-1) + n-1
- h(n-1) = h(n-2) + n-2
- ...
- h(4) = h(3) + 3
- h(3) = h(2) + 2
- h(2) = h(1) + 1
- H(1) = 0

In summary

- The number of handshakes is
- h(n) = h(1) + 1 + 2 +3 + 4 + ... + n-1
  - The sum of integer from 1 to n-1
  - Or equivalently n(n-1) / 2

What did we just do ?

- We defined the problem in terms of the problem itself
- We've been solving handshake problems but with smaller and smaller size
- Until we find problem small enough to know the answer instantly
- And we combine the result to subproblem for the result to the original problem

Recursion

- Divide and conquer approach for solving seemingly complicated problem
- By defining problem in terms of the problem itself but with smaller size
- Until we reach the case where the answer is trivial
- The final result is the combination of answer to the subproblems

Factorial problem

- What is 6! ?
  - 6! = 6 * 5 * 4 * 3 * 2 * 1
- Or we could write
  - 6! = 6 * 5!
- In general we have
  - n! = 1 (if n == 1)
  - n! = n * (n-1)! (if n is larger than 1)

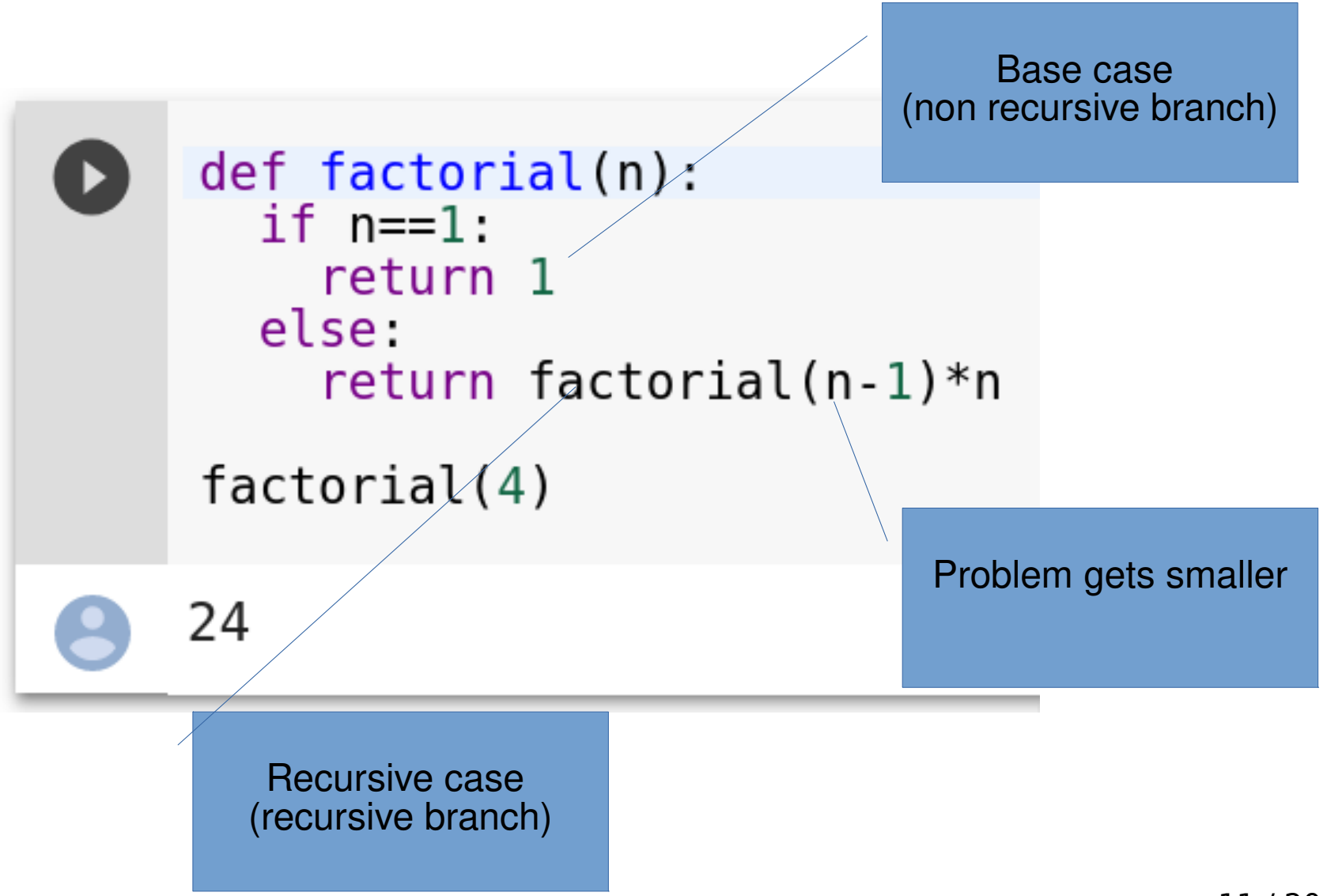
Recursive function for factorial

```
def factorial(n):
  if n==1:
    return 1
  else:
    return factorial(n-1)*n

factorial(4)
```

```
24
```

The recusive function



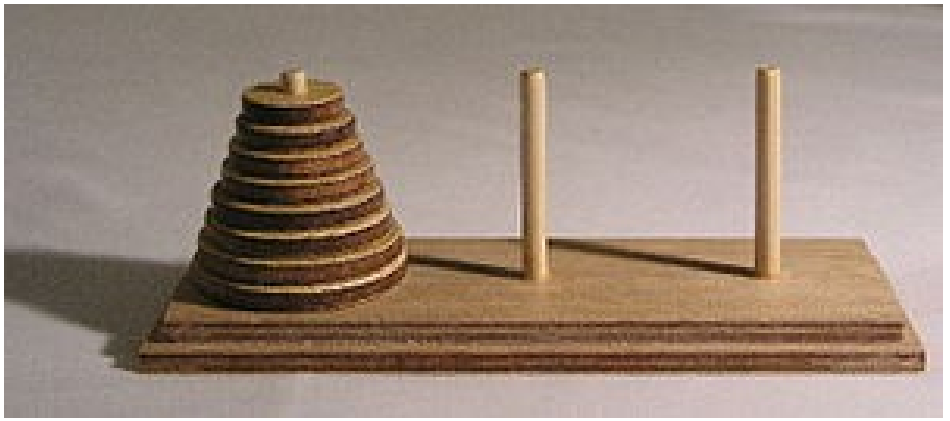
Common mistakes

- If there’s no base case (non recursive branch) the calculation will go on forever
  - Make sure your solution has base case
  - Make sure your code has non-recursive branch
- The problem has to get smaller and smaller everytime as we recursively call the function
  - This will eventually lead to base case

Why use recursion ?

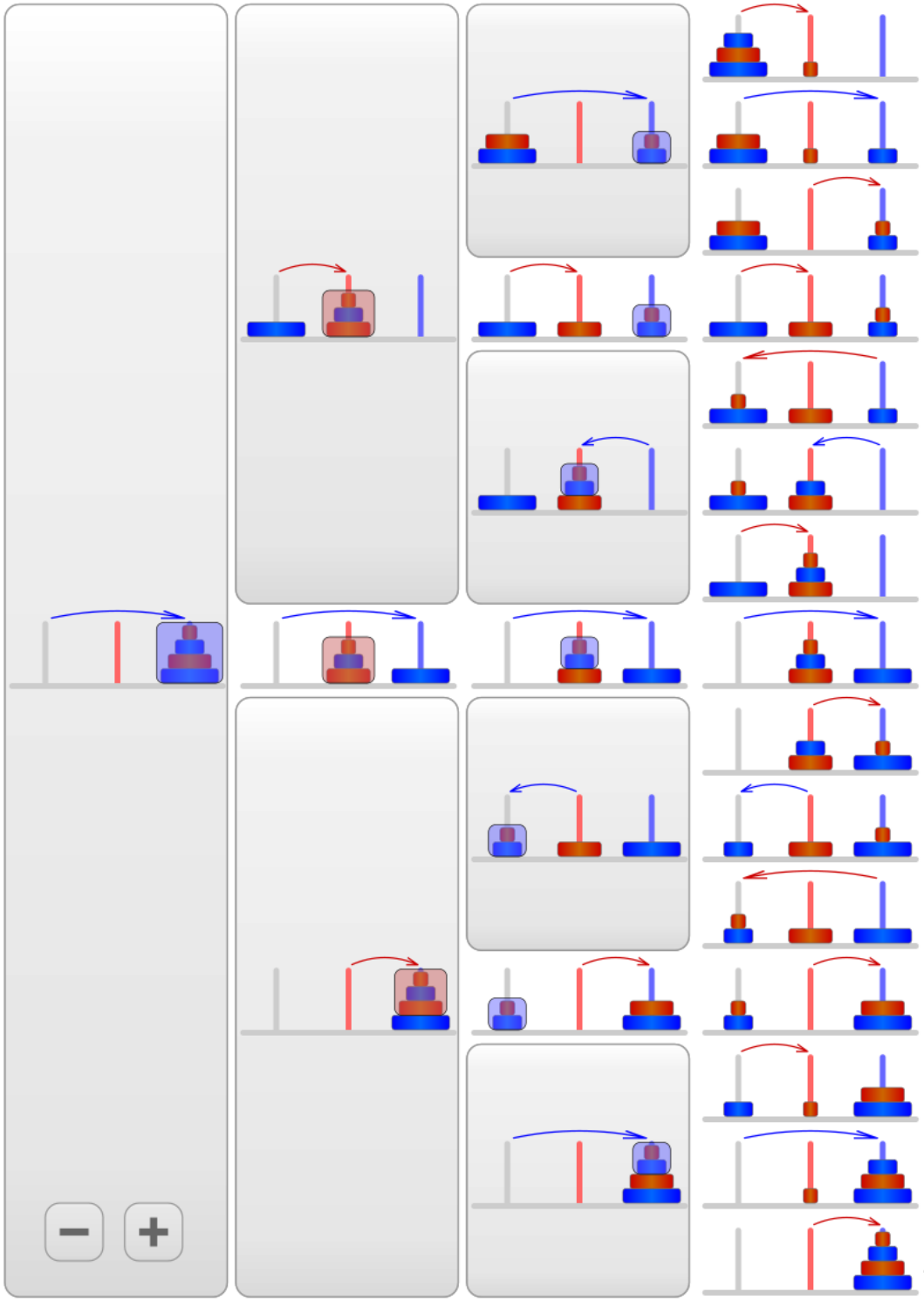
- Pros
  - Elegent code
  - Solve complex problem more easily
- Cons
  - Recursive function consumes more time and memory

Example 1: Tower of Hanoi



- Task: Move stack of discs to target peg
- Conditions
  - Only one disc could be moved at a time
  - A larger disc must never be stacked above a smaller one
  - One and only one extra peg could be used for intermediate storage of discs

Recursive solution



Exercise 2: Range sum

- Given a range defined by two numbers find the sum of all numbers in the range

```
03 def range_sum(lo, hi):
04     if (lo == hi):
05         return ____________________________
06     else:
07         return ____________________________
08
09 print range_sum(10, 15)  # 75
```

Exercise 3: Power

- Find the power of base to the exponent exp

```
02 def power(base, exp):
03     # assume exp is non-negative integer
04     if (exp == 0):
05         return ____________________________
06     else:
07         return ____________________________
08
09 print power(2, 5)  # 32
```

Exercise 4: Sequence

- พจน์ที่ k ของ Sequence a มี Definition ดังนี้

$$a_k = \begin{cases} 2, & k = 1 \\ a_{k-1} + 2k, & k > 1 \end{cases}$$

- ให้เขียนฟังก์ชัน Recursive `term_k(`k`)` เพื่อคำนวณค่าพจน์ a_k

```
02 def term_k(k):
03     if k == 1:
04         return _____________________________
05     else
06         return _____________________________
```

Exercise 5: Digit Sum

- Given an integer, find the summation of its digits

```
08 def digit_sum(n):
09     if ____________________________:
10         return ____________________________
11     else:
12         return ____________________________
13
14 print(digit_sum(1027))    # 10
```

Remember: The Three Laws of Recursion

1) A recursive algorithm must have a base case.
2) A recursive algorithm must change its state and move toward the base case.
3) A recursive algorithm must call itself, recursively.