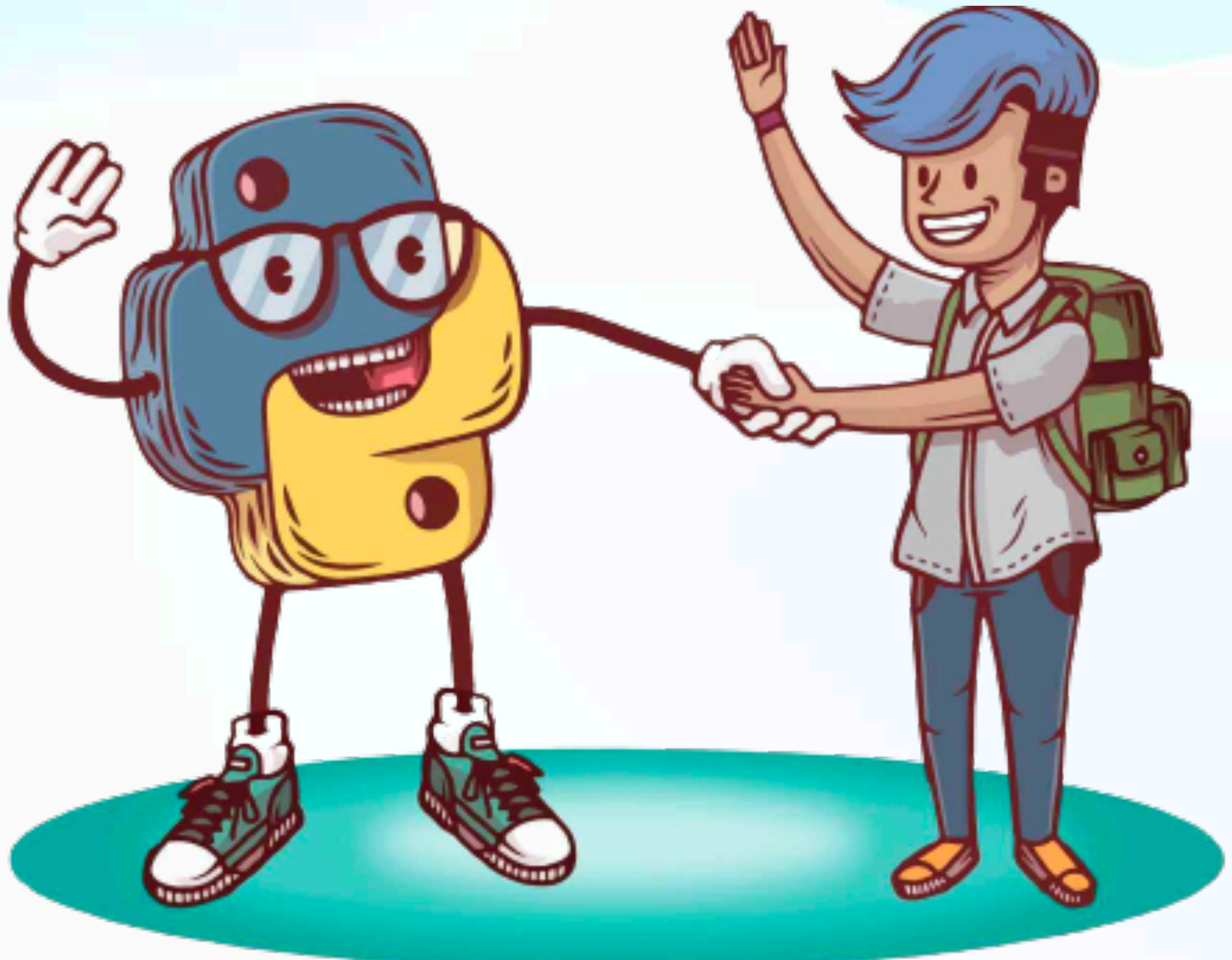
Recitation 2

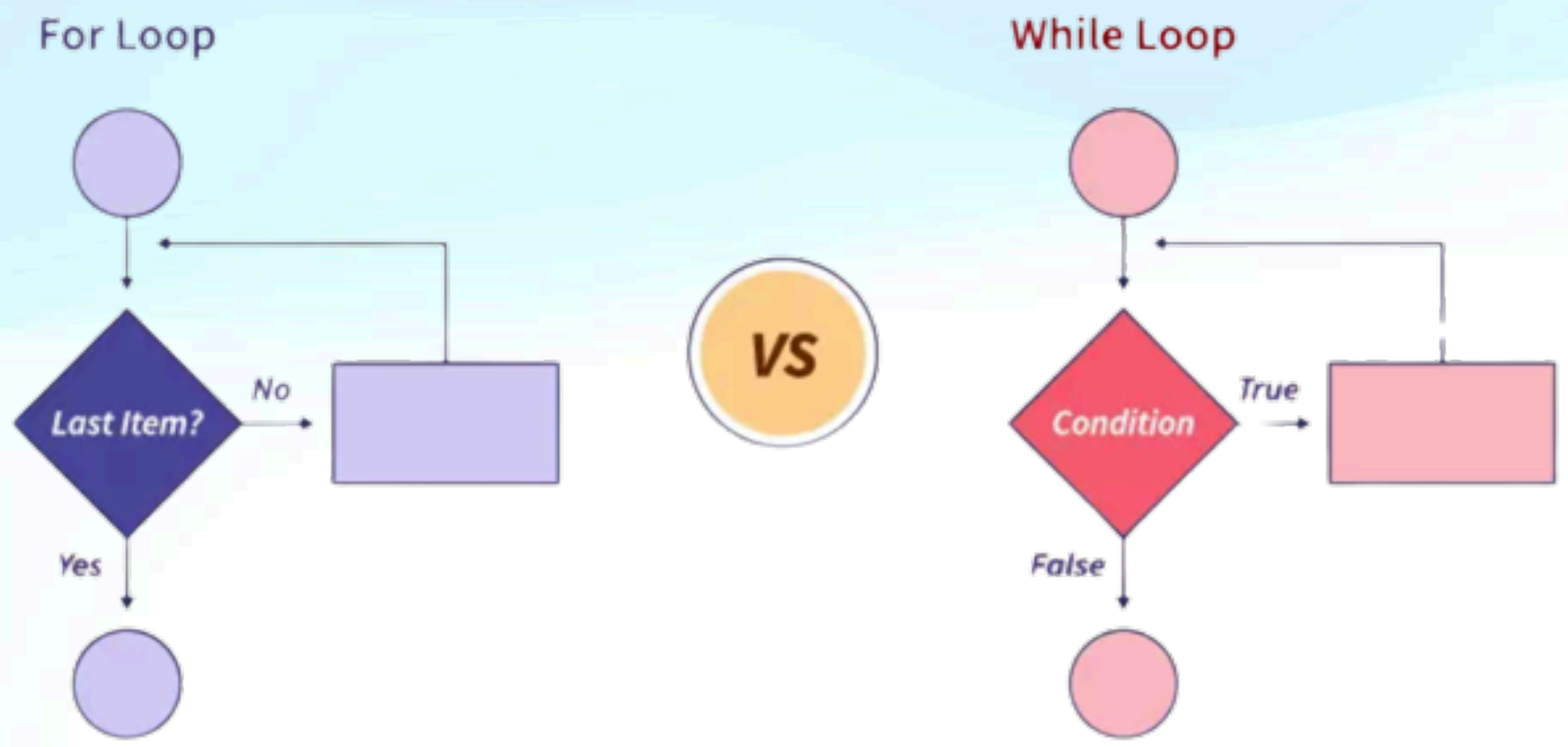
CS 210



Anamika Lochab

For vs While

For loops and **While** loops are **generally interchangeable** in terms of their ability to perform repeated tasks, but they are often used in **different scenarios** because they offer **different advantages** or are more readable for certain tasks.



Colab

Functions

Functions

- ***def*** creates a function and assigns it a name
- ***return*** sends a result back to the caller
- Arguments are passed by assignment
- Arguments and return types are not declared

```
def <name>(arg1, arg2, ..., argN):
  <statements>
  return <value>
```

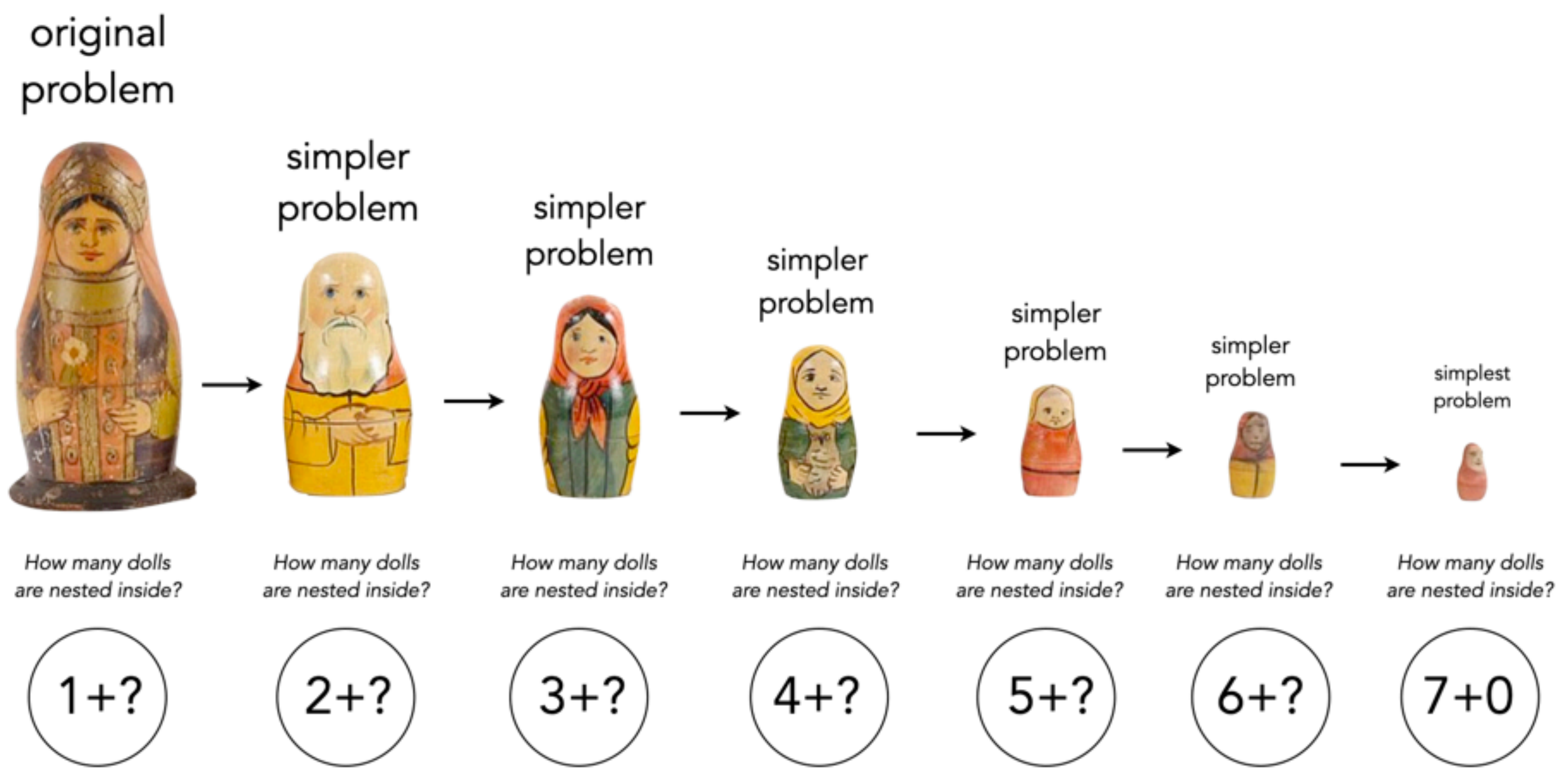
Functions

- All functions in Python have a **return** value
  - even if no return line inside the code.
- Functions **without a return returns the special value None**.
- There is **no function overloading** in Python.
  - Two different functions **can't have the same name, even if they have different arguments.**
- Functions **can be used as any other data type**. They can be:
  - Arguments to function
  - Return values of functions
  - Assigned to variables
  - Parts of tuples, lists etc.

Recursion

Recursion is the process by which a function calls itself directly or indirectly.

- Base case(s) (non-recursive) - One or more simple cases that can be done right away.
- Recursive case(s) - One or more cases that require solving “simpler” version(s) of the original problem. • By “simpler” , we mean “smaller” or “shorter” or “closer to the base case”.

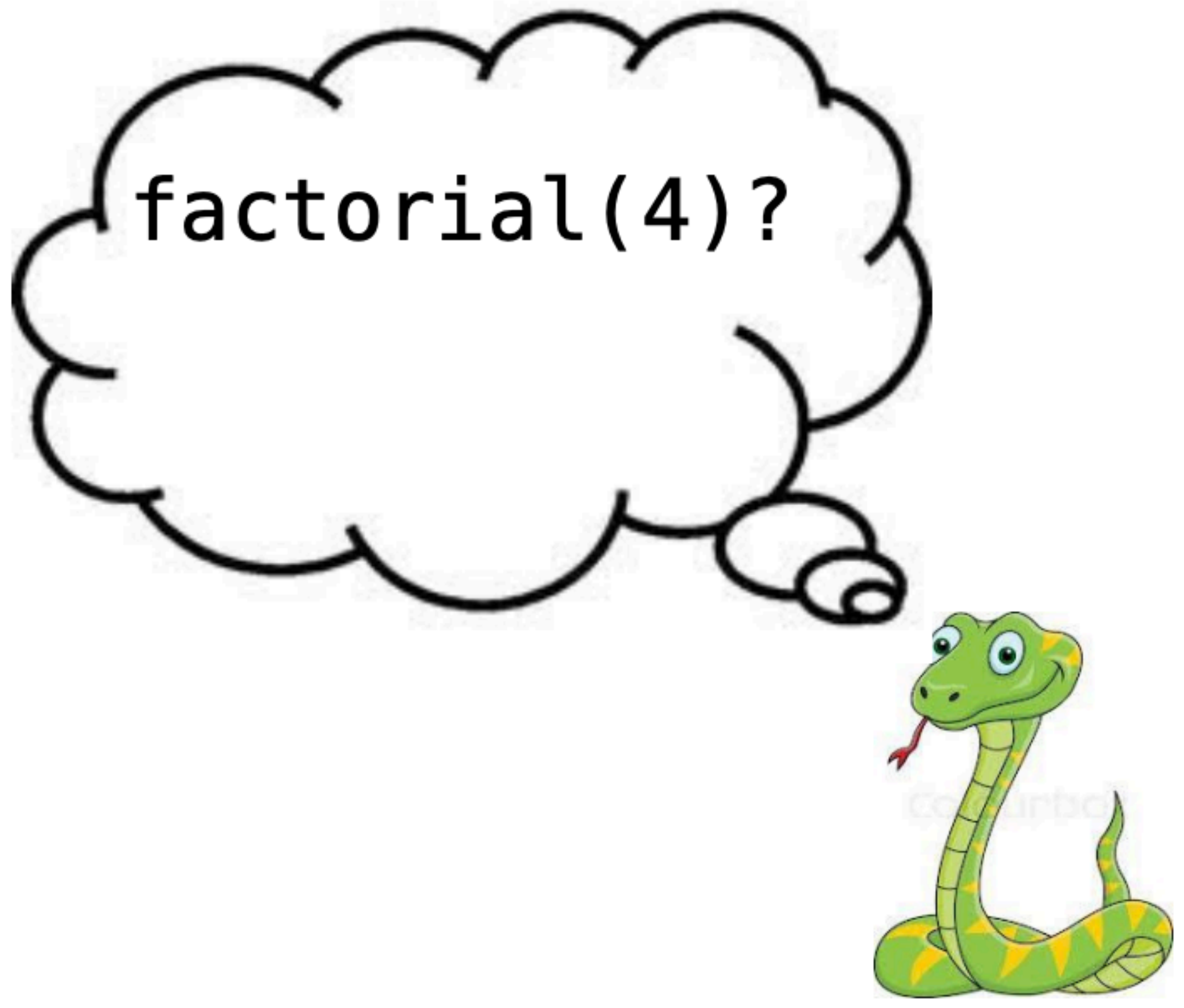

original problem
simpler problem
simpler problem
simpler problem
simpler problem
simpler problem
simplest problem
How many dolls are nested inside?
How many dolls are nested inside?
How many dolls are nested inside?
How many dolls are nested inside?
How many dolls are nested inside?
How many dolls are nested inside?
How many dolls are nested inside?
1+?
2+?
3+?
4+?
5+?
6+?
7+0

Colab

Inside Python Recursion

S
T
A
C
K

n=4



Inside Python Recursion

S

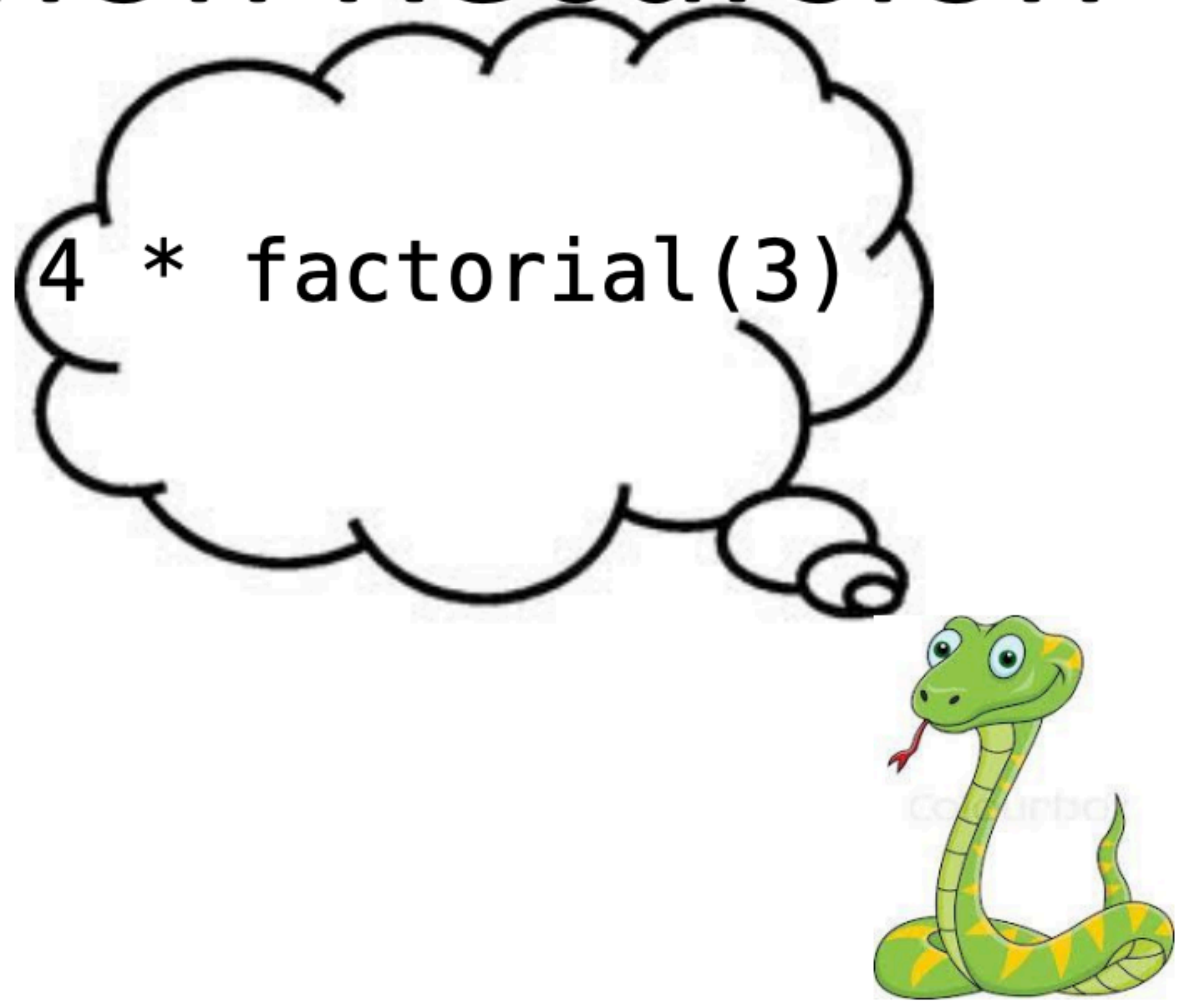
T

A

C

K

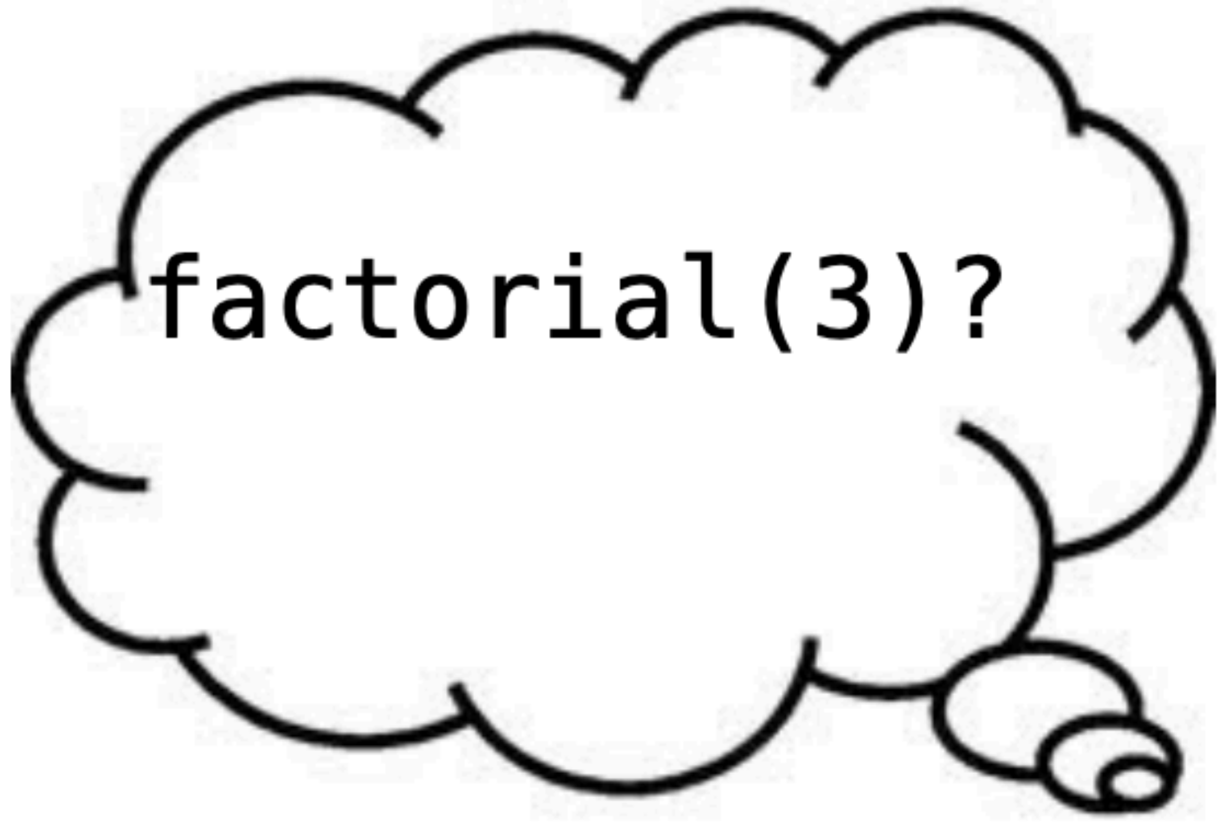
n=4   `factorial(4)? = 4 * factorial(3)`



Inside Python Recursion

S n=4 `factorial(4)? = 4 * factorial(3)`

T n=3 `factorial(3)?`



A

C

K

Inside Python Recursion

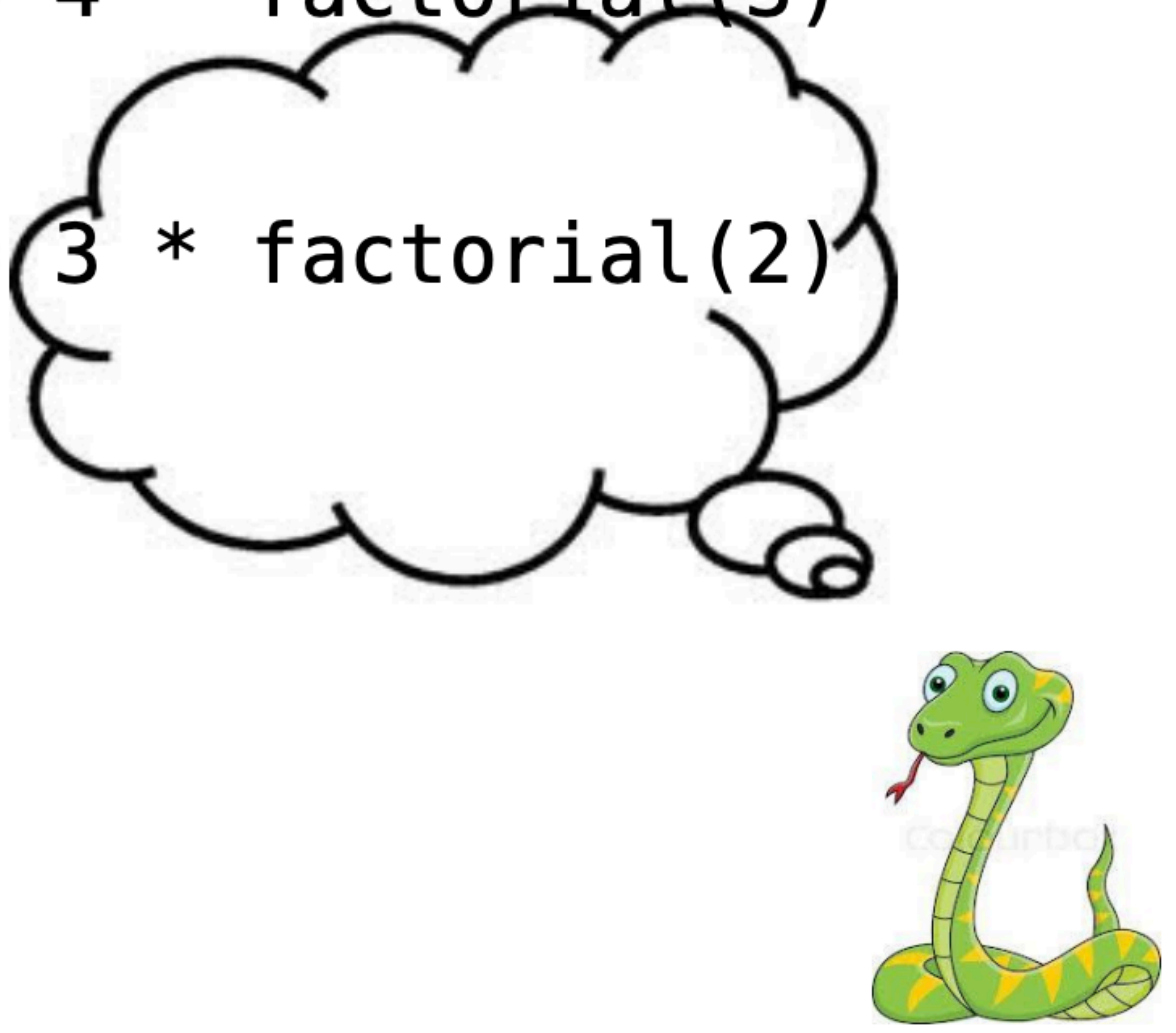
S n=4 `factorial(4)? = 4 * factorial(3)`

T n=3 `factorial(3)? = 3 * factorial(2)`

A

C

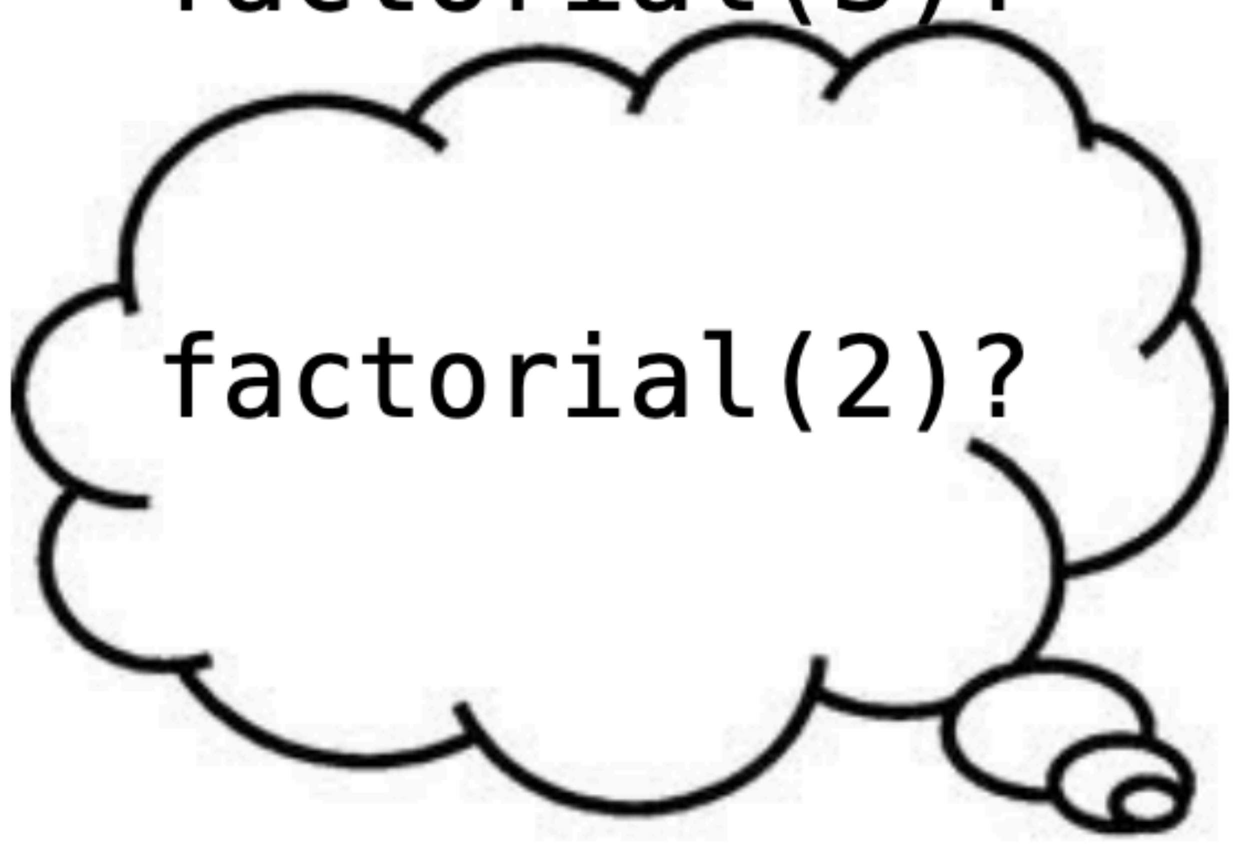
K



Inside Python Recursion

S n=4 `factorial(4)? = 4 * factorial(3)`

T n=3 `factorial(3)? = 3 * factorial(2)`



A n=2 `factorial(2)?`

C

K

Inside Python Recursion

S

T

A

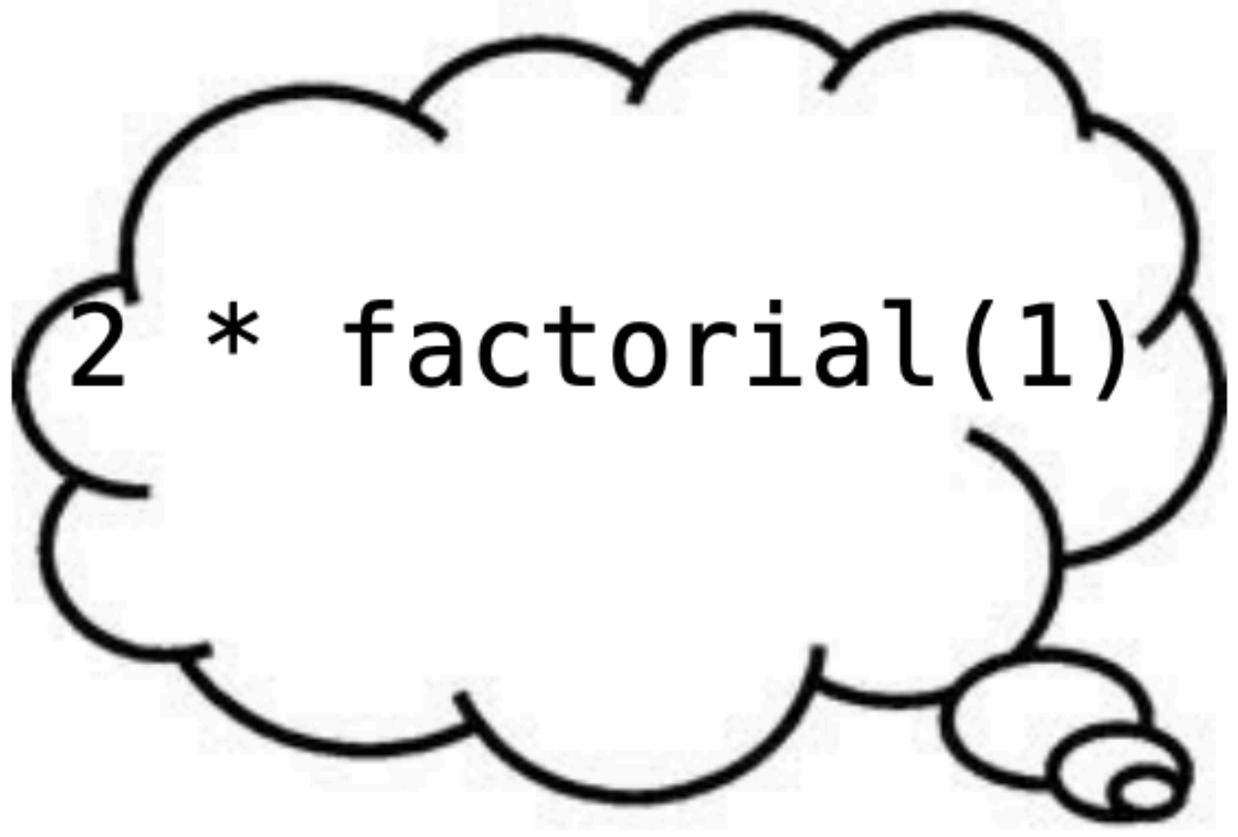
C

K

```
n=4     factorial(4)? = 4 * factorial(3)

n=3     factorial(3)? = 3 * factorial(2)

n=2     factorial(2)? = 2 * factorial(1)
```



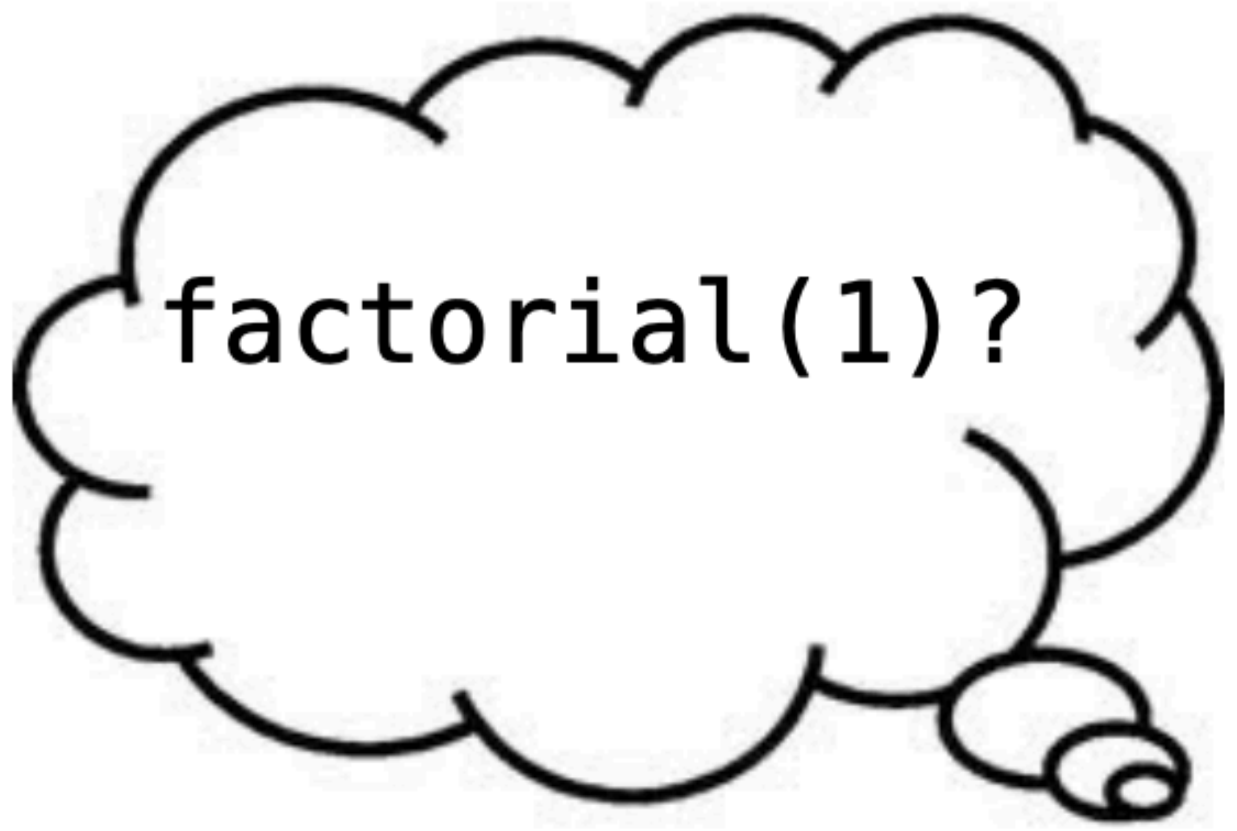
Inside Python Recursion

S n=4 `factorial(4)? = 4 * factorial(3)`

T n=3 `factorial(3)? = 3 * factorial(2)`

A n=2 `factorial(2)? = 2 * factorial(1)`

C n=1 `factorial(1)?`



K

Inside Python Recursion

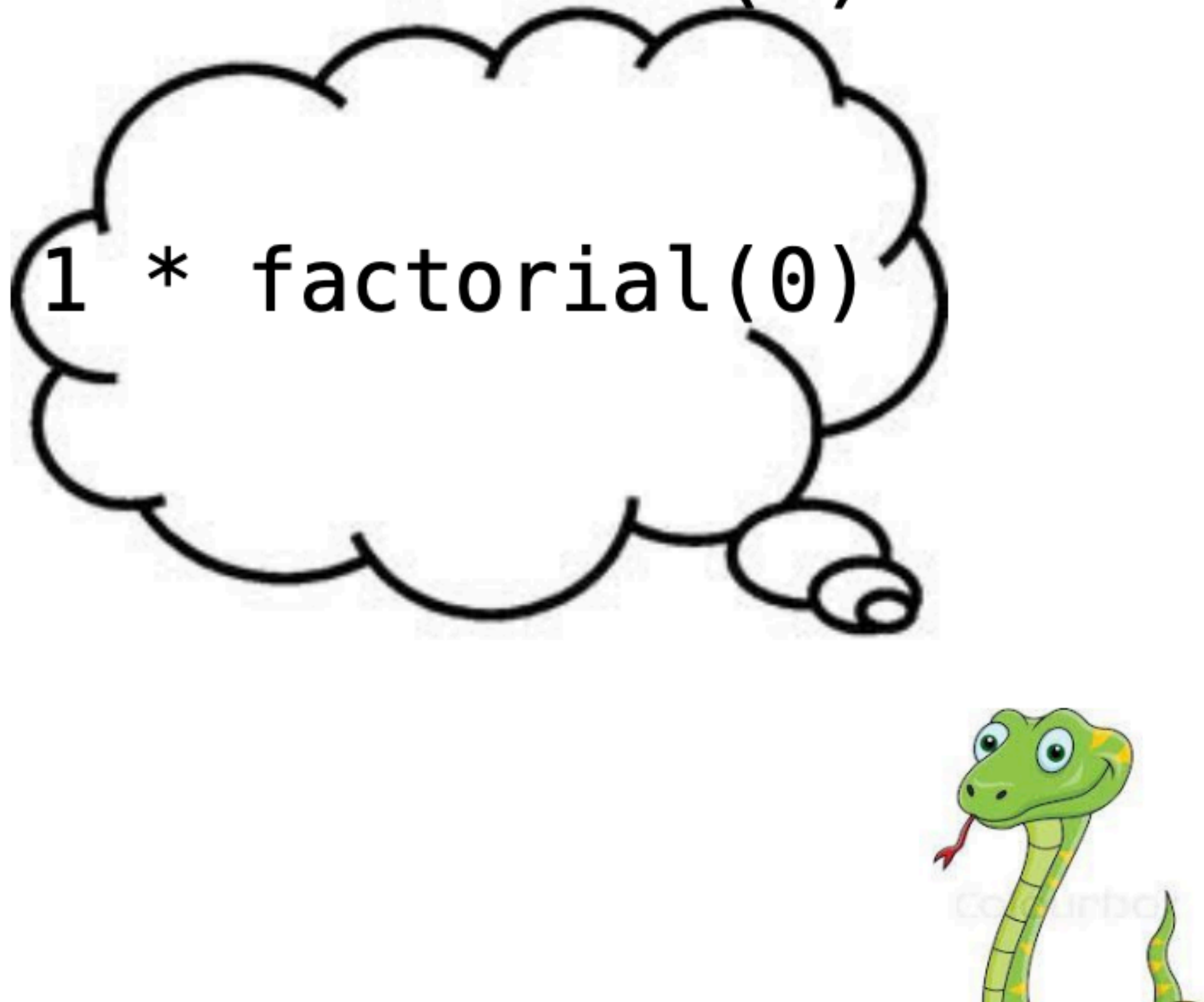
S n=4 `factorial(4)? = 4 * factorial(3)`

T n=3 `factorial(3)? = 3 * factorial(2)`

A n=2 `factorial(2)? = 2 * factorial(1)`

C n=1 `factorial(1)? = 1 * factorial(0)`

K



Inside Python Recursion

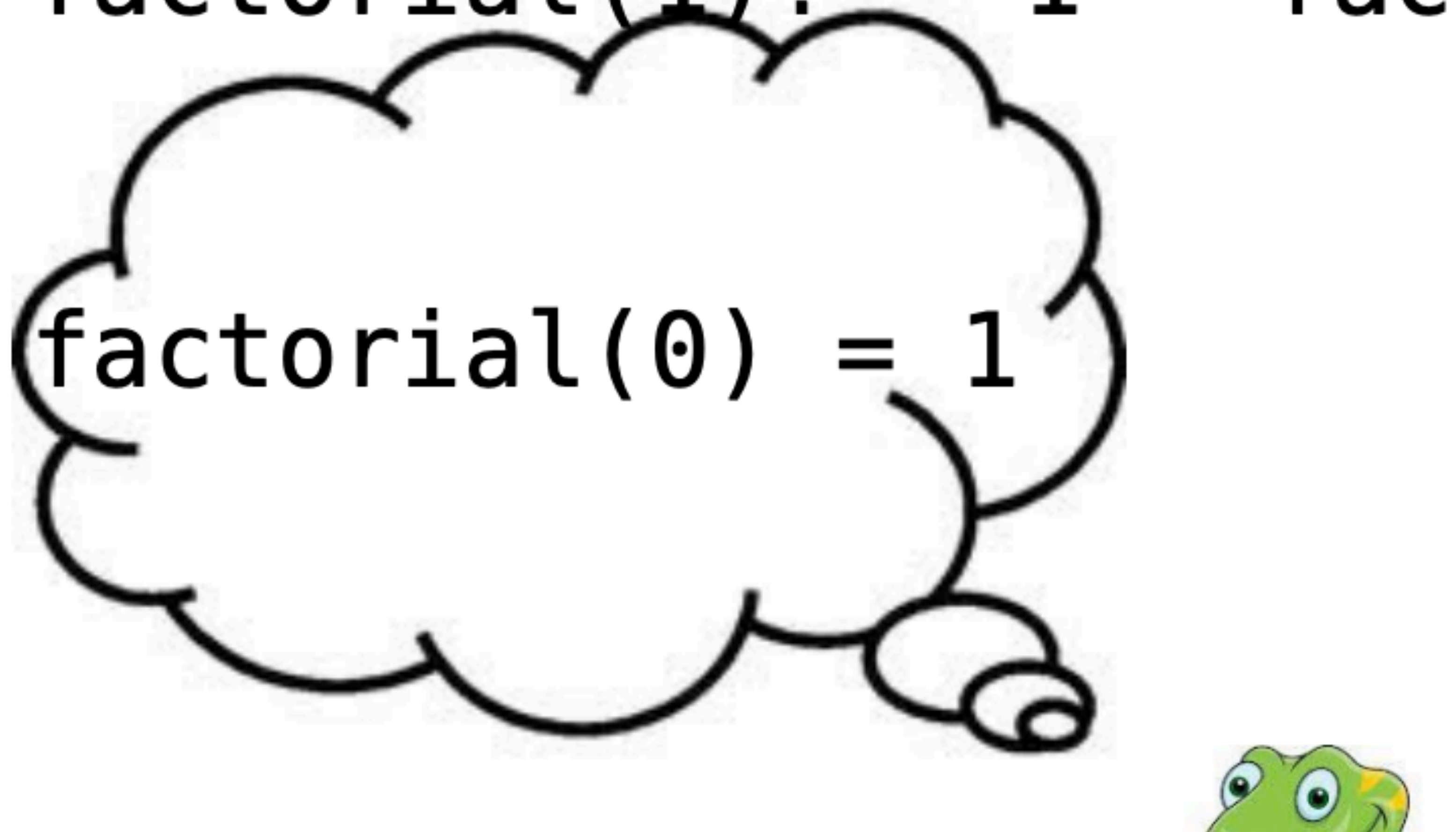
S n=4 `factorial(4)? = 4 * factorial(3)`

T n=3 `factorial(3)? = 3 * factorial(2)`

A n=2 `factorial(2)? = 2 * factorial(1)`

C n=1 `factorial(1)? = 1 * factorial(0)`

K n=0 `factorial(0) = 1`



Inside Python Recursion

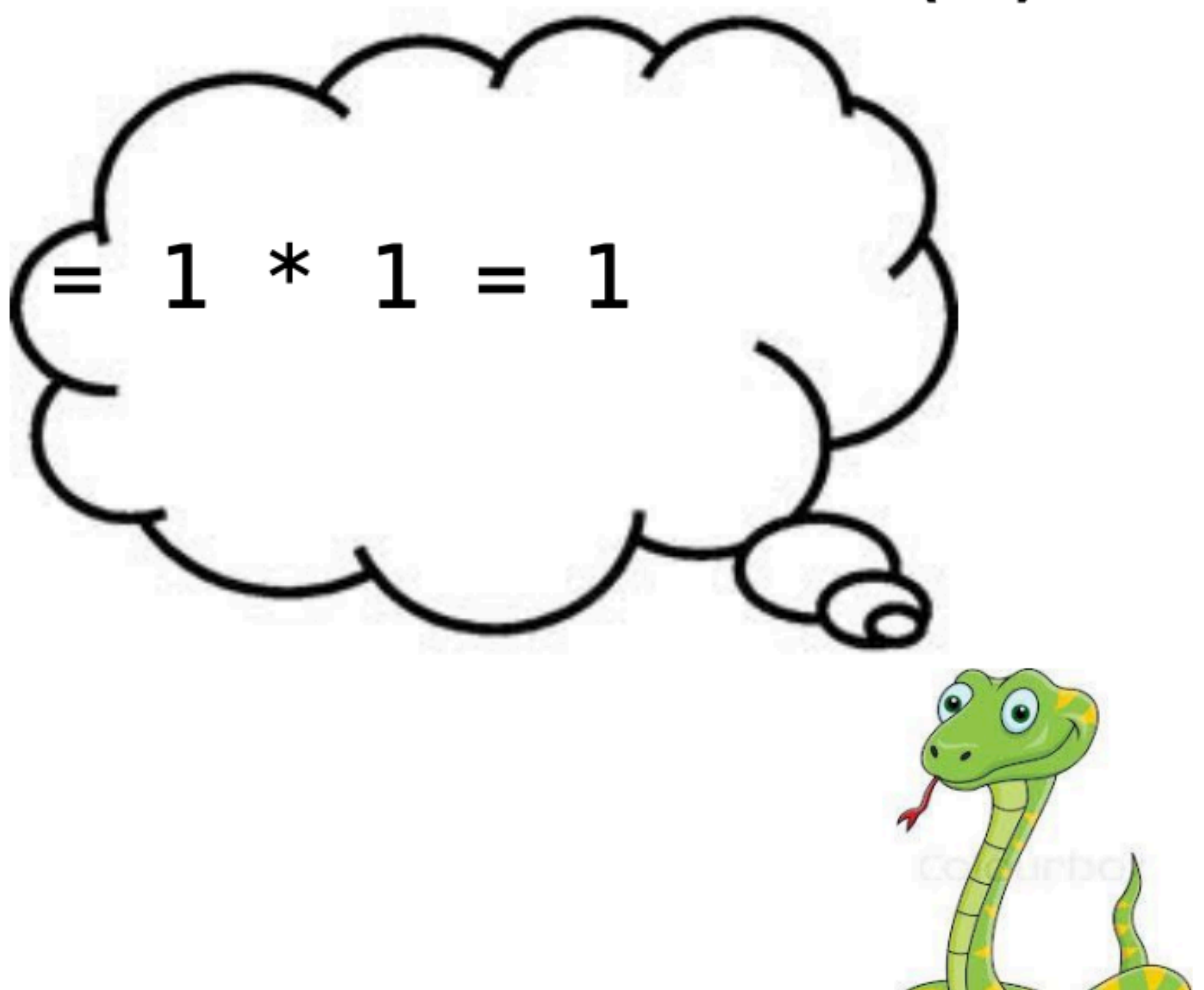
S n=4 factorial(4)? = 4 * factorial(3)

T n=3 factorial(3)? = 3 * factorial(2)

A n=2 factorial(2)? = 2 * factorial(1)

C n=1 factorial(1) = 1 * 1 = 1

K



Inside Python Recursion

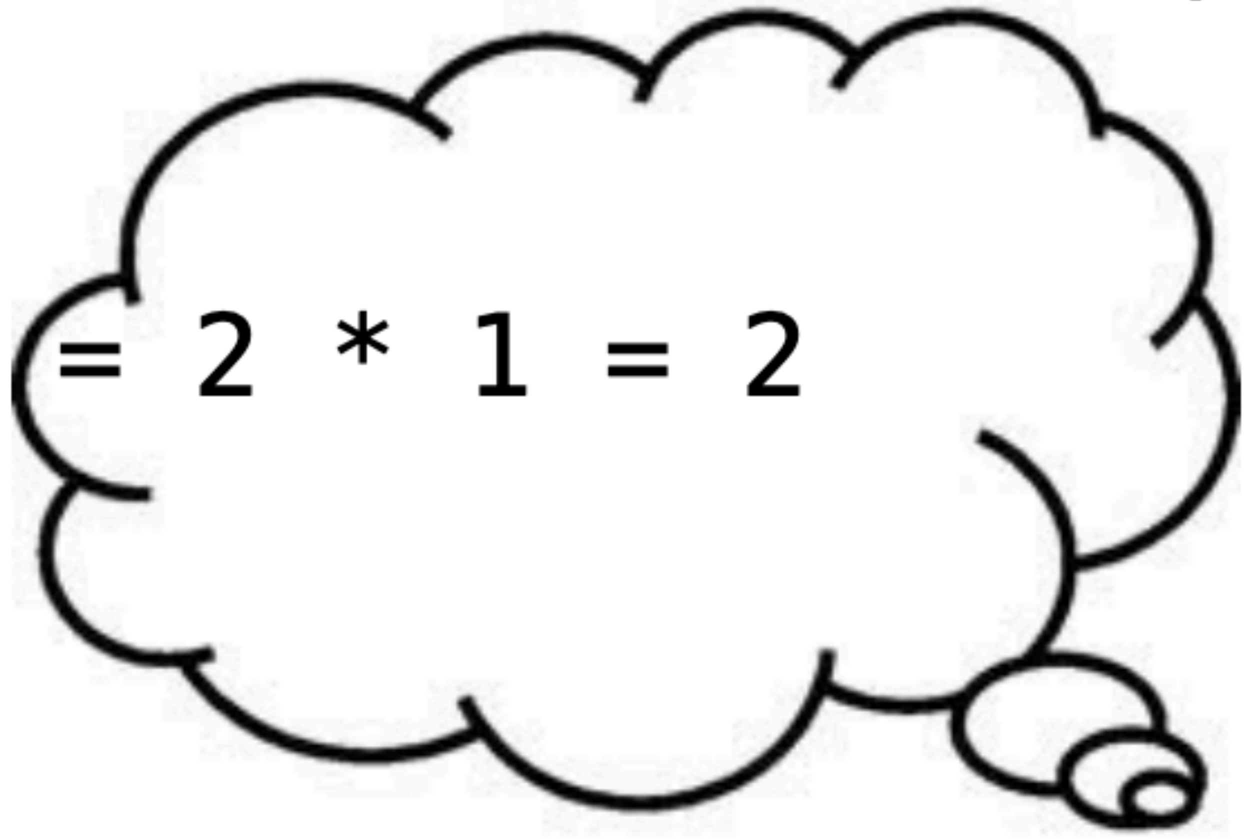
S n=4 factorial(4)? = 4 * factorial(3)

T n=3 factorial(3)? = 3 * factorial(2)

A n=2 factorial(2) = 2 * 1 = 2

C

K

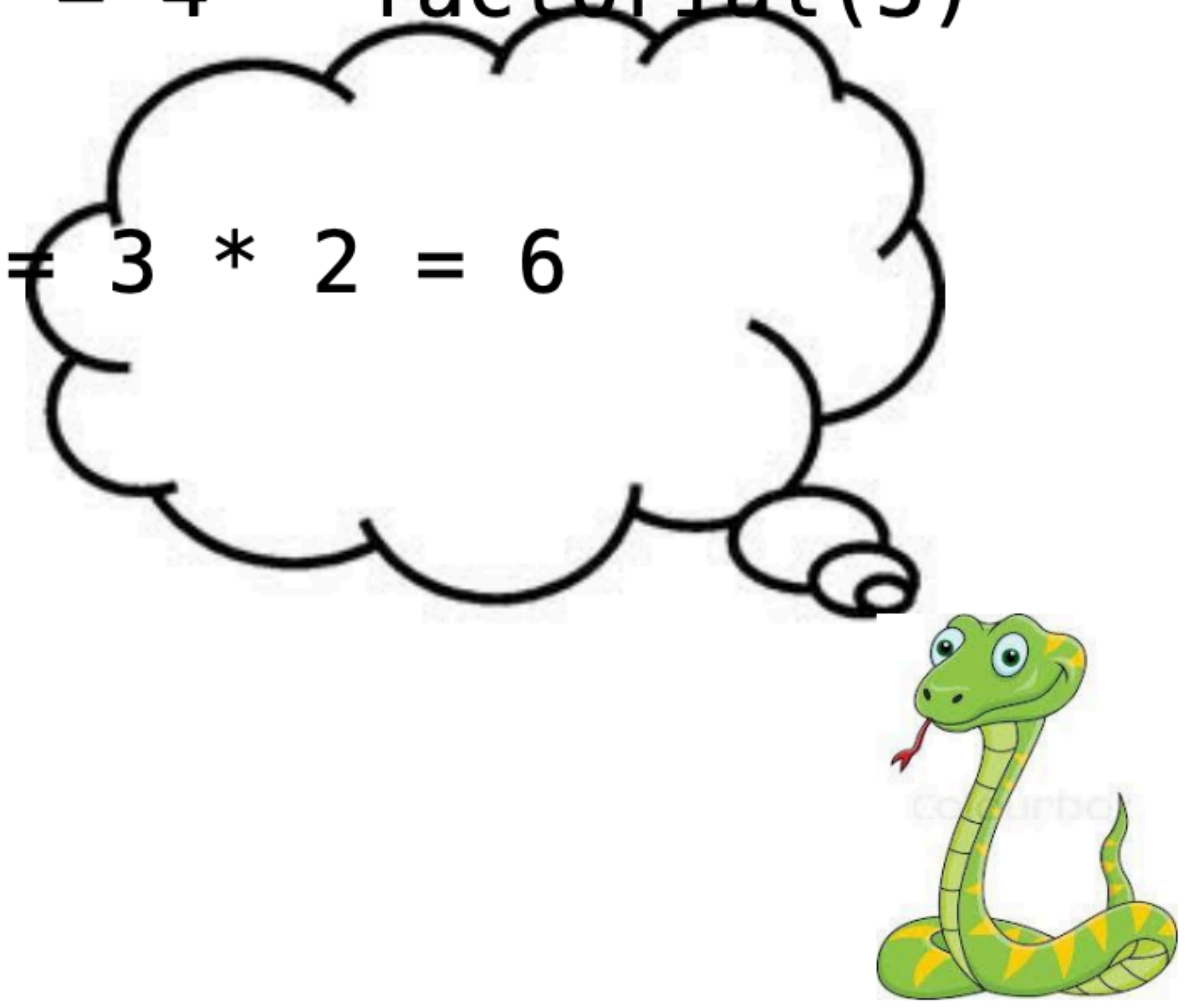


Inside Python Recursion

S
T
A
C
K

n=4 factorial(4)? = 4 * factorial(3)

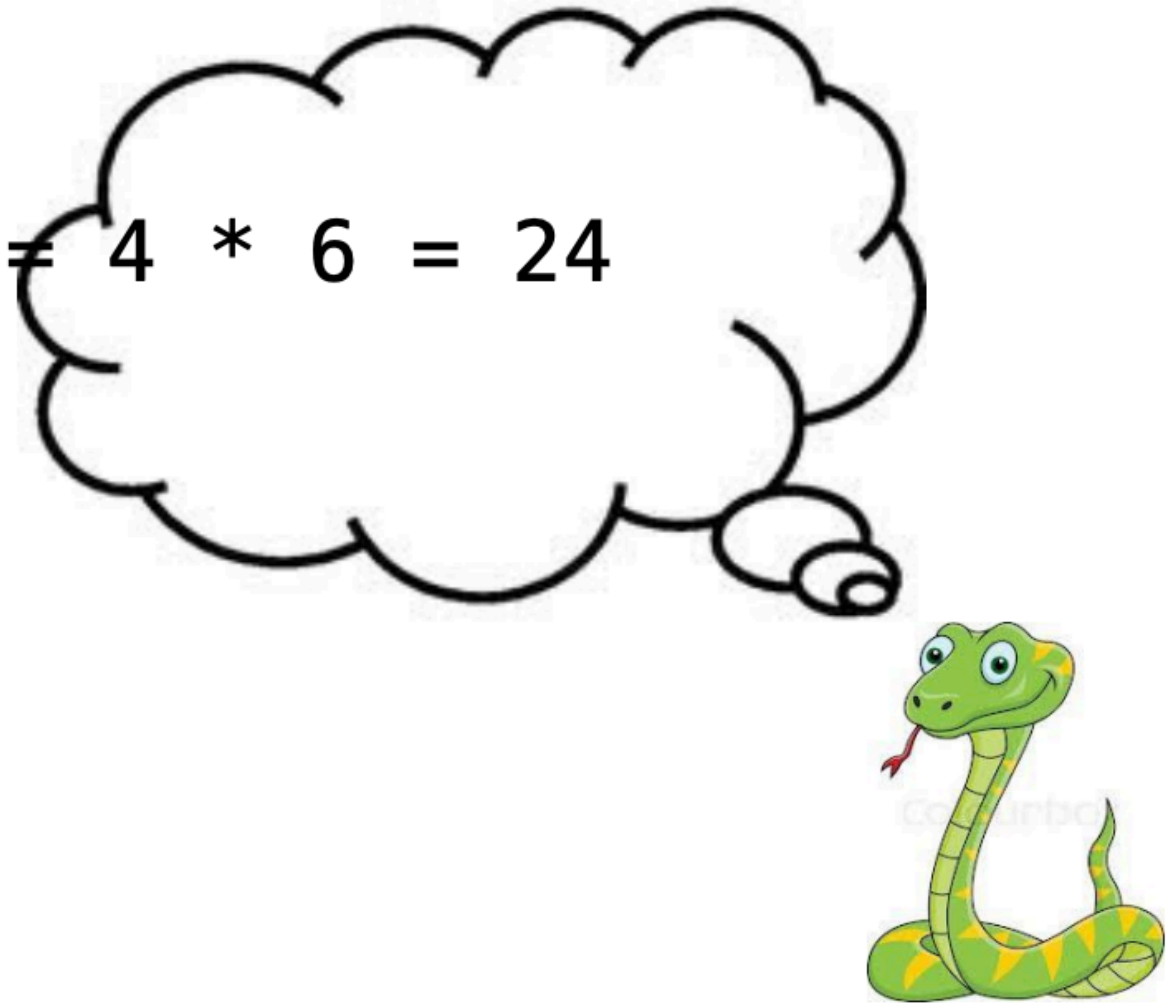
n=3 factorial(3) = 3 * 2 = 6



Inside Python Recursion

S
T
A
C
K

n=4 factorial(4) = 4 * 6 = 24



List Comprehension

Different Ways to Express a List

- **Explicit Definition**: The most basic way to create a list is to manually define it.

```
my_list = [1, 2, 3, 4, 5]
```

- **Using for loop:** Starting with an empty list and adding elements using append().

```
my_list = []
for i in range(1, 6):
    my_list.append(i)
```

- **List Comprehension:** It provides a more concise way to create lists.

```
my_list = [i for i in range(1, 6)]
```

Colab

Greg the frog is trying to jump to the other side of a pond. The frog starts out excited and makes a big leap, but each subsequent leap is of variable length. The frog keeps leaping until it crosses the pond.

The shortest path from his position to the other side is 30 feet. The first leap is 5 feet. Model this scenario to notify when the frog has crossed the pond.

Greg is a smart frog, he doesn't divert from the straight line (the shortest path to the other side).

Also create a dictionary to store attributes of the frog's journey i.e, current position, leaps, number of leaps, pond length.