COMP10001

2023 Sem 1

Foundations of Computing

Week 11

Tutorial/Workshop

Today's Tutorial

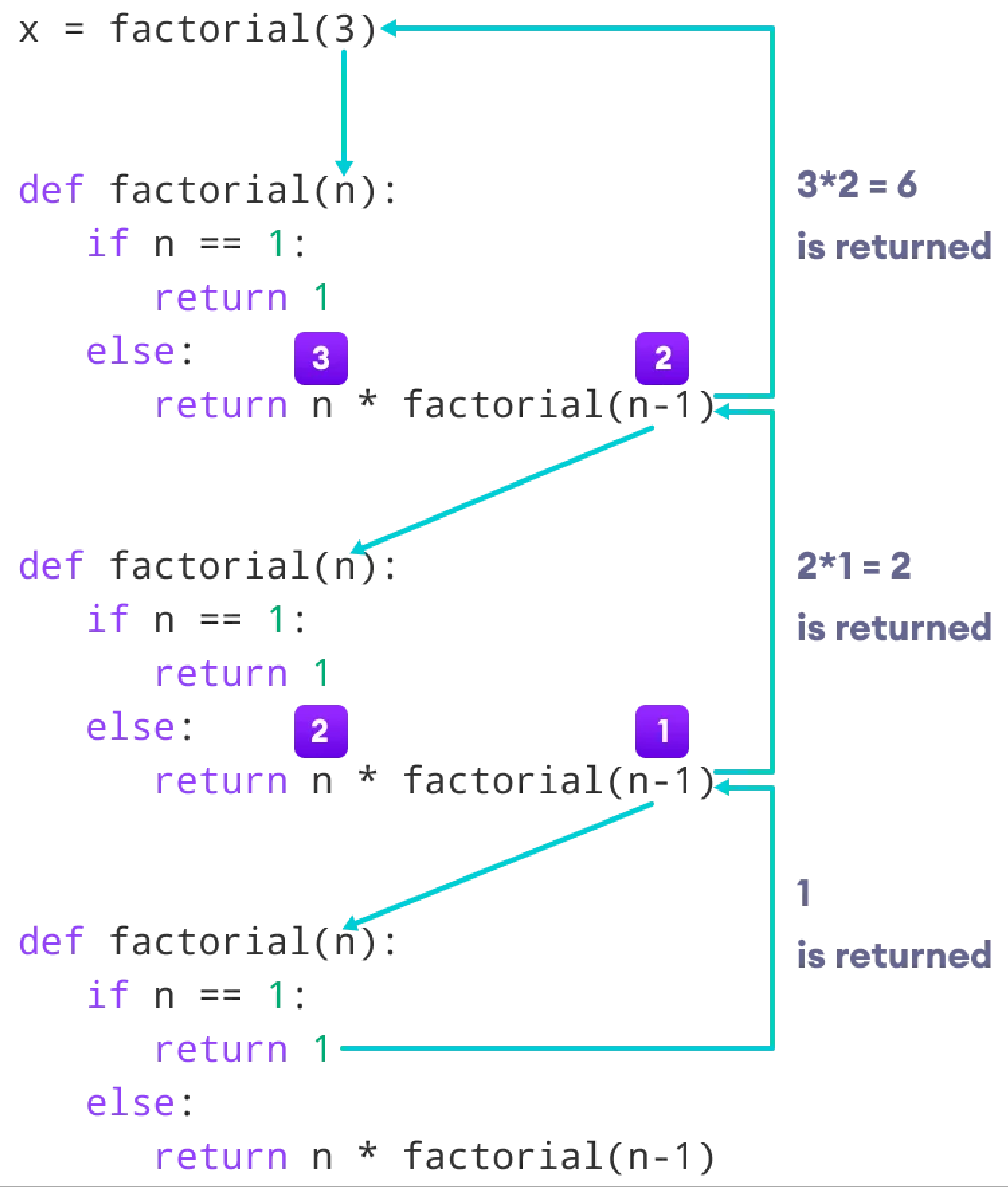
1. Recursion
2. Algorithms

Recursion

Function calls itself repeatedly, as opposed to a loop

Generally the output gets smaller until it reaches the answer

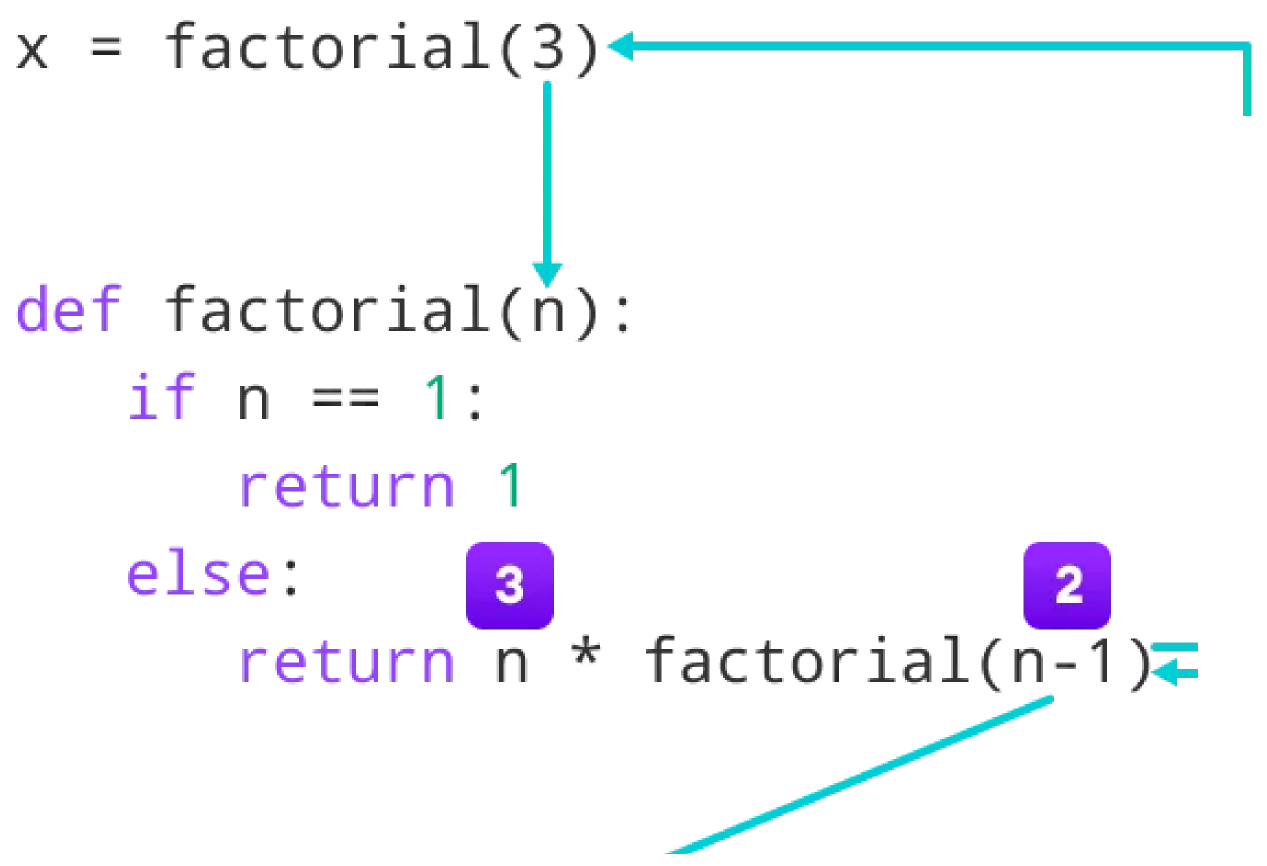
```
def factorial(x):
    if x == 1:          Base case
        return x
    else:
                                     Recursive case
        return x*factorial(x-1)
```



```
x = factorial(3)

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

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

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

```
x = factorial(3)
```

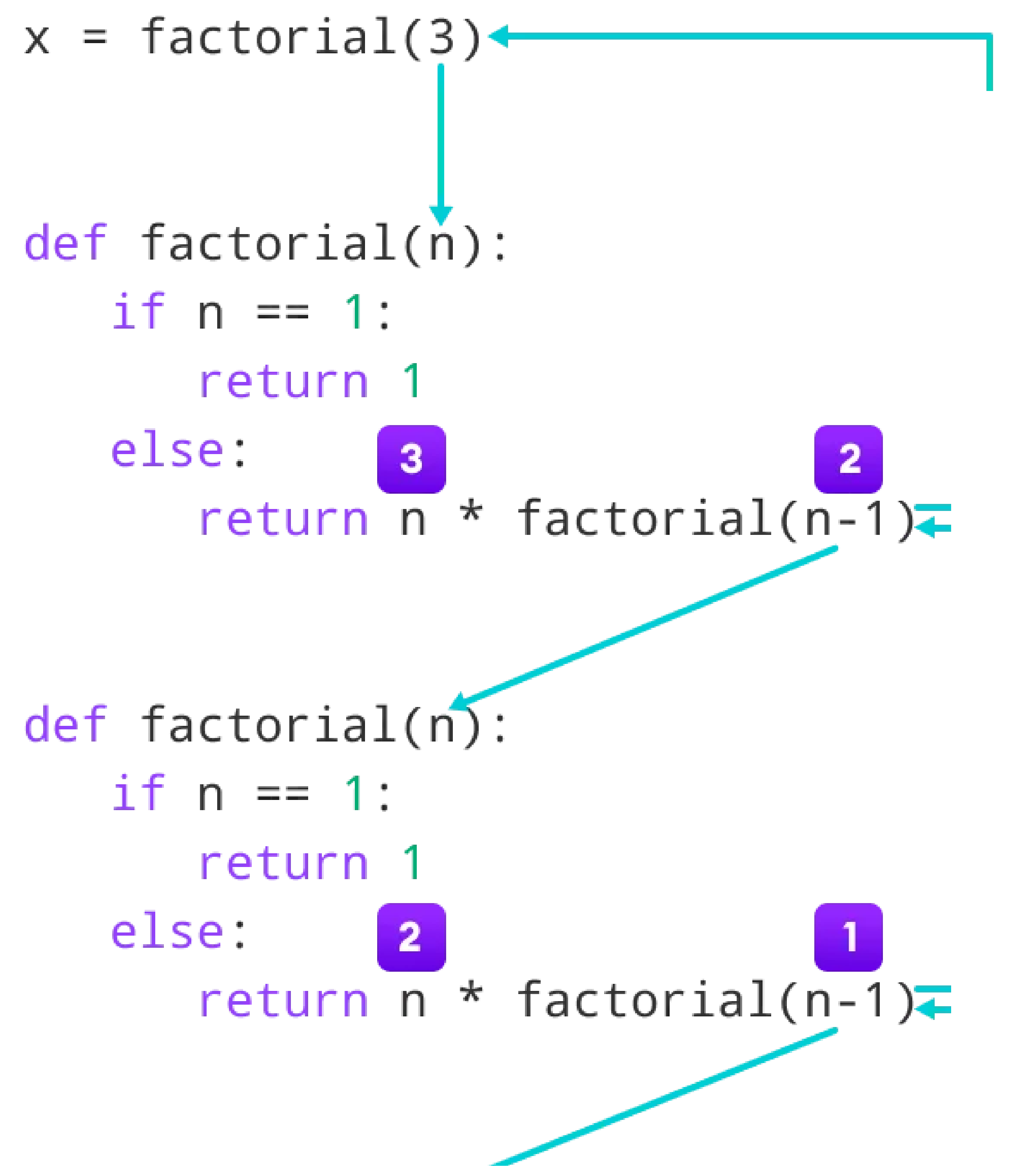
factorial(3)



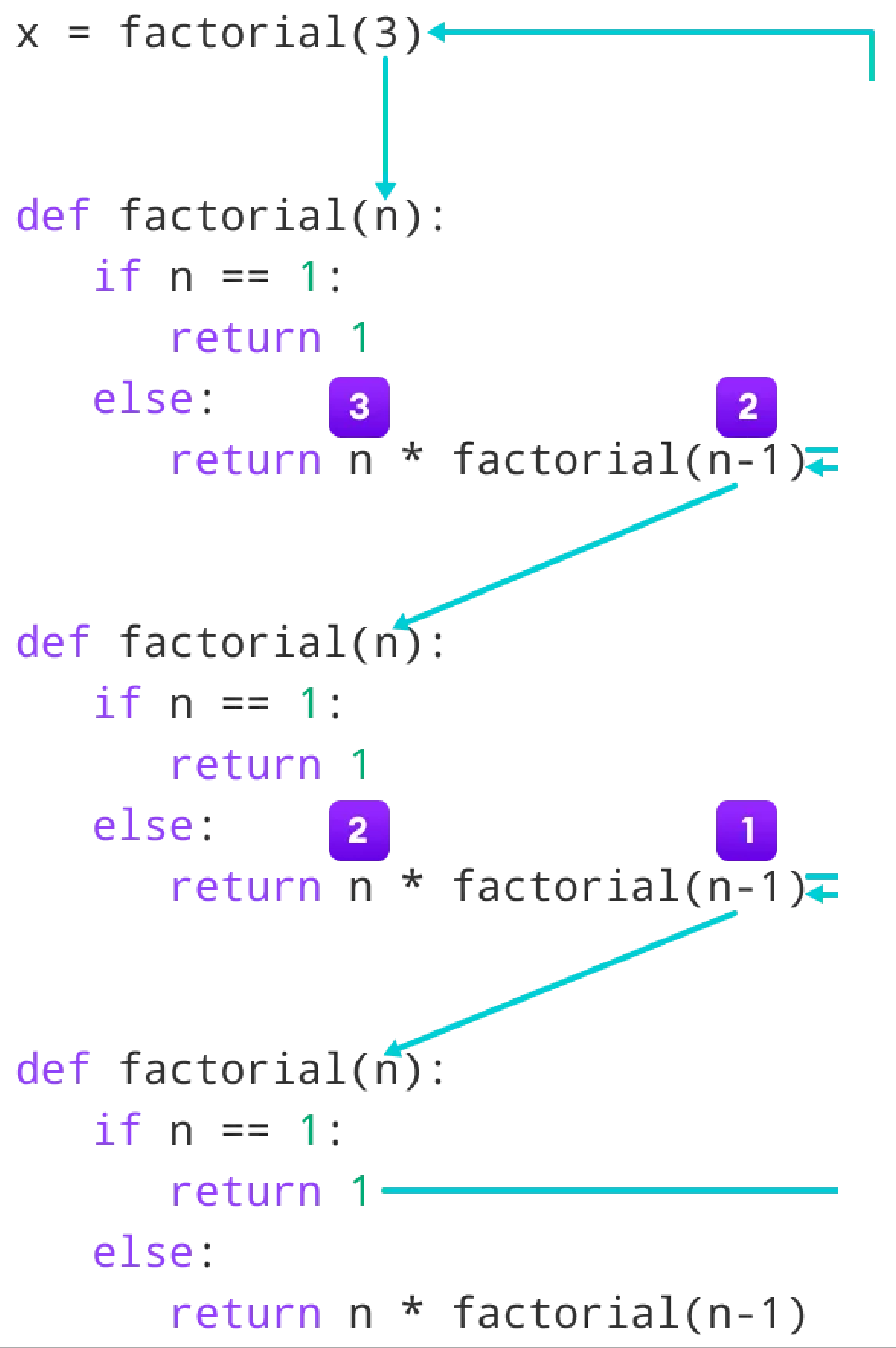
```
x = factorial(3)


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

factorial(3)

factorial(2)



```
x = factorial(3)


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


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



```
x = factorial(3)
```

factorial(3)

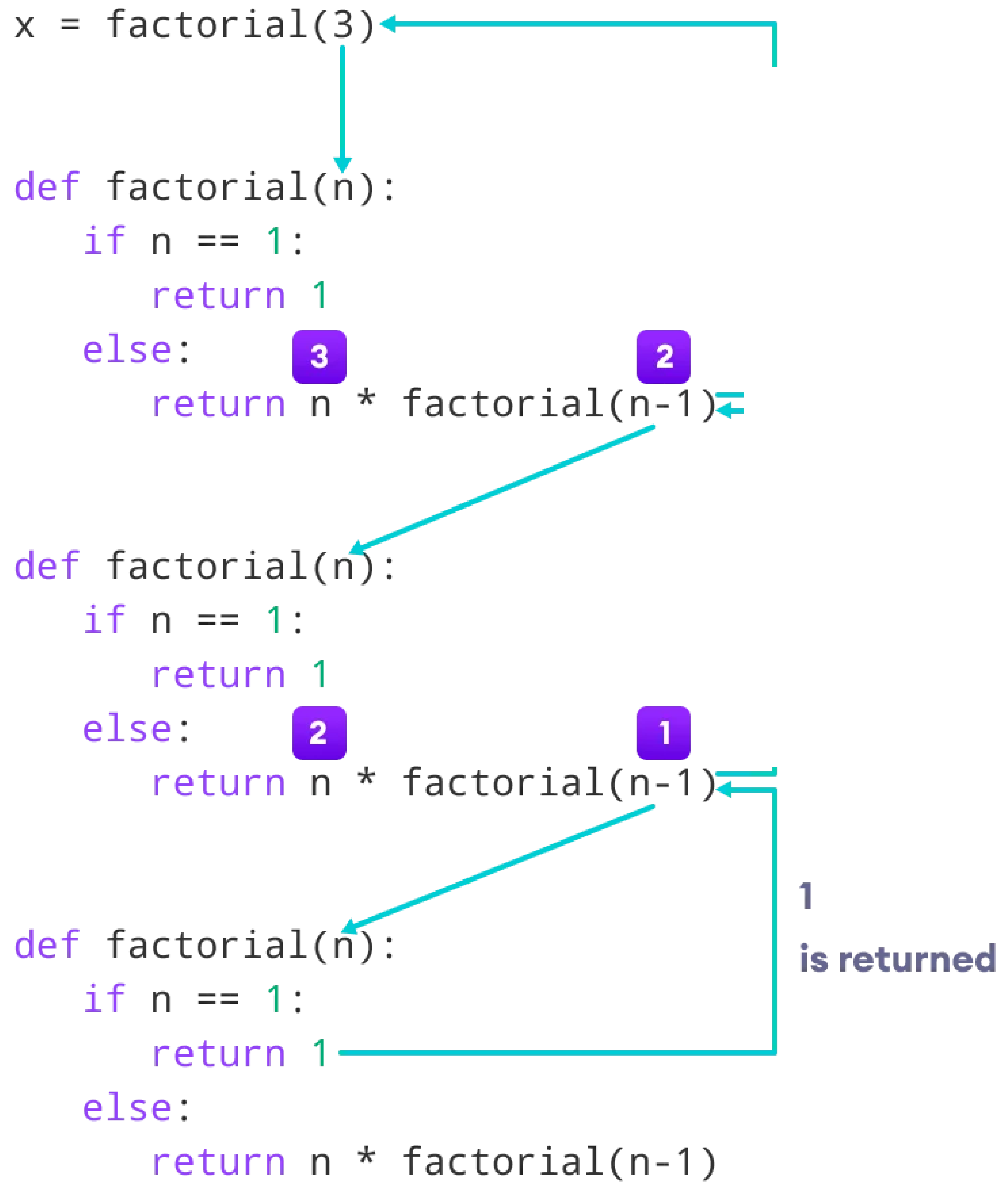
```
def factorial(n):
    if n == 1:
        return 1
    else:
        return n * factorial(n-1)
```

factorial(2)

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

factorial(1)

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



```
x = factorial(3)
```

factorial(3)

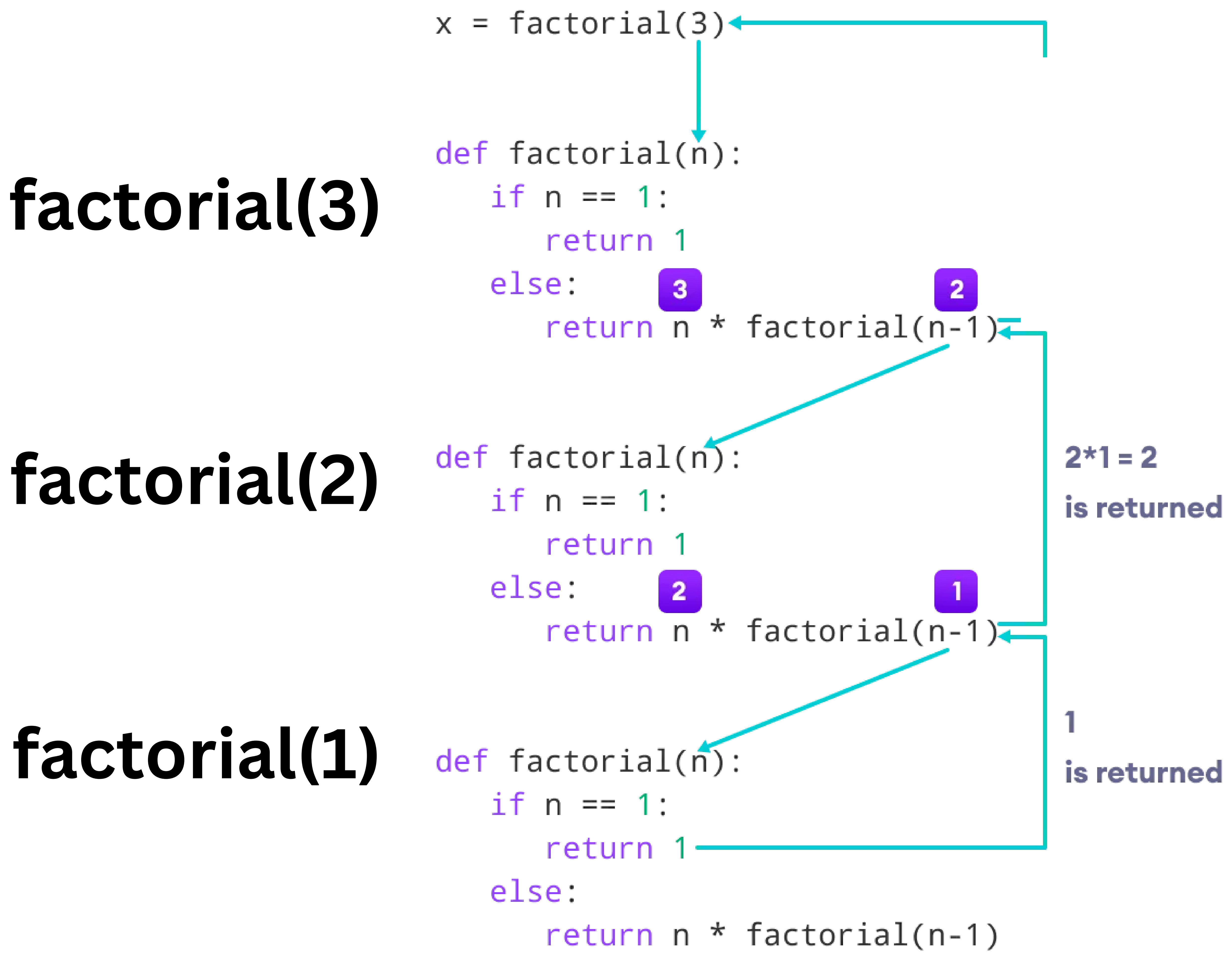
```
def factorial(n):
    if n == 1:
        return 1
    else:
        return n * factorial(n-1)
```

factorial(2)

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

factorial(1)

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



```
x = factorial(3)
```

factorial(3)

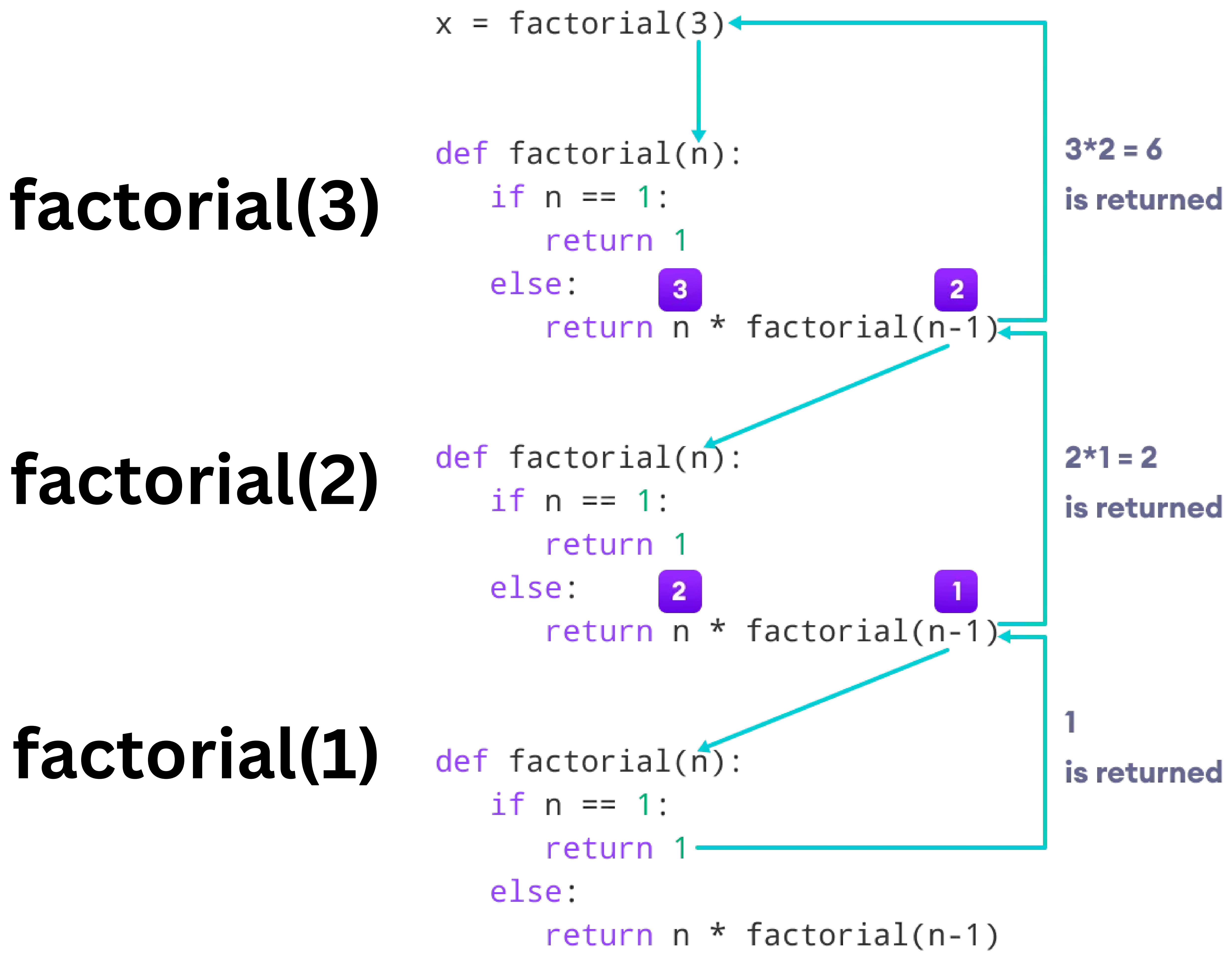
```
def factorial(n):
    if n == 1:
        return 1
    else:
        return n * factorial(n-1)
```

factorial(2)

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

factorial(1)

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



```python
x = factorial(3)

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

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

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

Recursion vs iteration

Useful when you would need to nest many, many loops

But if iteration can be done in reasonable number of loops, it is often better since it is less "expensive" than recursion

Iteration only goes 1 direction - can't pass information back
Recursion also allows many parameters to be passed in

Recursion is powerful for algorithms

COMP10001

2023 Sem 1

Exercises 1&2

Exercise 1 answer

(a)
```
def mystery(x):
    if len(x) == 1:
        return x[0]
    else:
        y = mystery(x[1:])
        if x[0] > y:
            return x[0]
        else:
            return y
```

A: *The* `if` *block is the base case, and the* `else` *block is the recursive case. The function returns the largest element in the list/tuple.*

(b)
```
def mistero(x):
    a = len(x)
    if a == 1:
        return x[0]
    else:
        y = mistero(x[a//2:])
        z = mistero(x[:a//2])
        if z > y:
            return z
        else:
            return y
```

A: *The* `if` *block is the base case, and the* `else` *block is the recursive case. Like (a), this function returns the largest element in the list/tuple. This function uses two recursive calls, while the first uses one. There's no difference in the calculated output.*

Exercise 2 answer

```
def can_make_change(amount, coins):
    # base case: success
    < blank 1 >
        return True

    # base case: failure
    if amount < 0 or len(coins) == 0:
        < blank 2 >

    # recursive case: handle two possibilities, either:
    # 1. another of this coin value gets used, or
    # 2. we don't need another coin of this value
    coin = coins[-1]
    return (can_make_change(< blank 3 >, coins)
            or can_make_change(amount, coins[:-1]))
```

A: *Blank 1:* `if amount == 0:`

Blank 2: `return False`

Blank 3: `amount - coin`

Algorithms

A sequence of steps for solving an instance of a particular problem type

We consider **correctness** and **efficiency**

Correctness: does it give the right output?

Efficiency: how fast? how much storage? how much processing power?

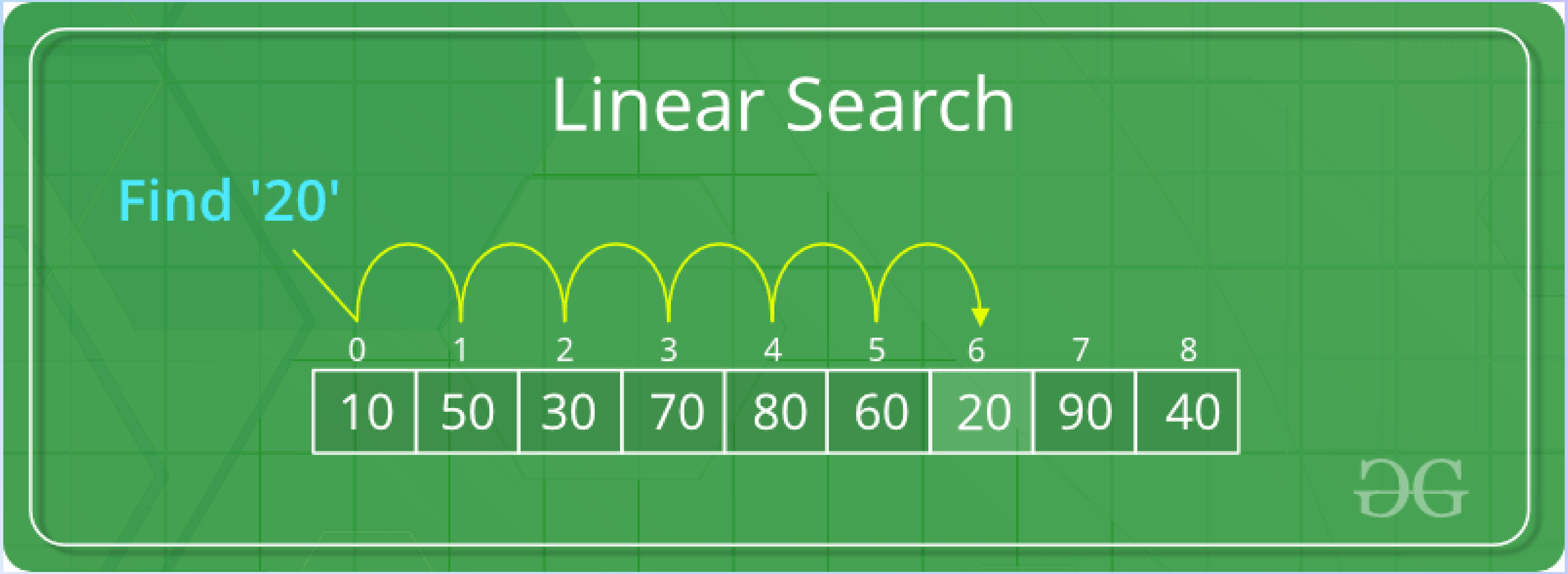
Algorithms

Exact vs **approximate**

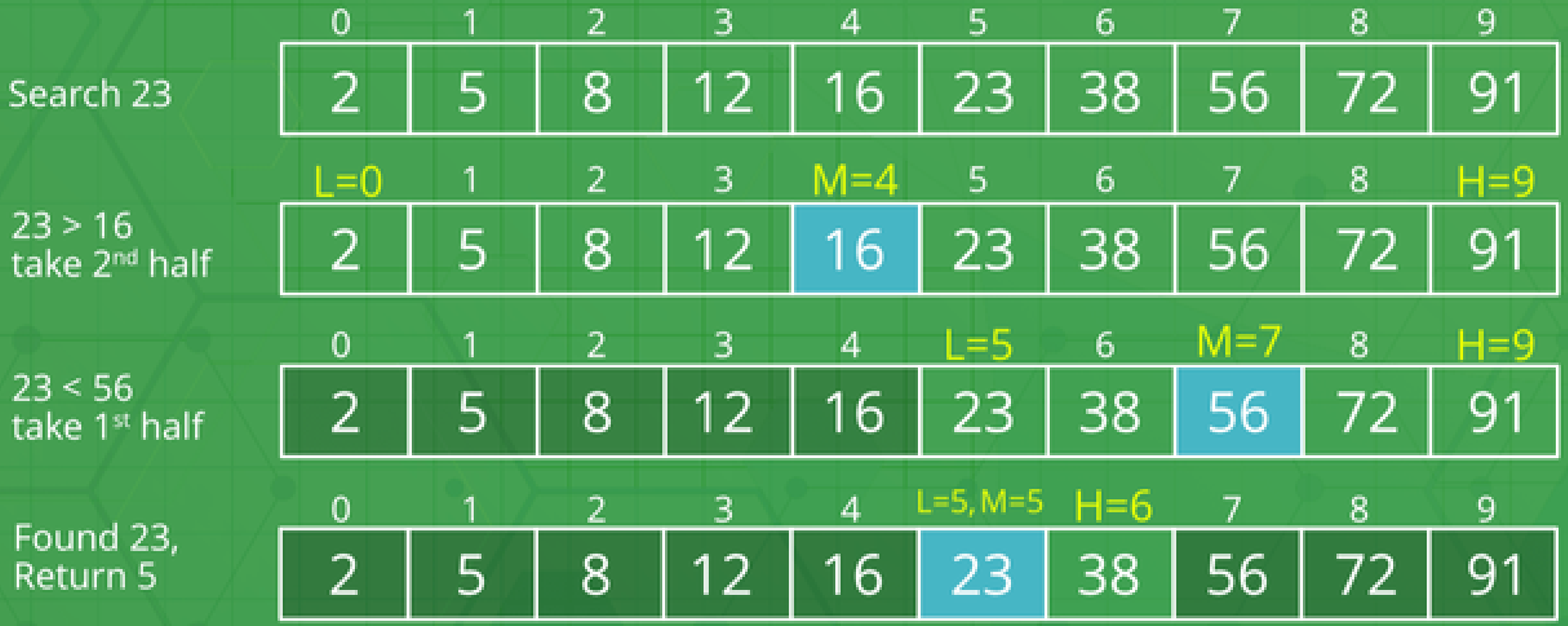
Exact: high correctness, but possibly low efficiency

Approximate: high efficiency, but possibly low correctness

Good for different use cases


Linear Search
Find '20'
0
1
2
3
4
5
6
7
8
10
50
30
70
80
60
20
90
40

Binary Search



Algorithms – exact or approximate?

Brute force

Heuristic

Simulation

Divide and conquer

Algorithms – exact or approximate?

Brute force
Exact - find every answer and test them 1 by 1

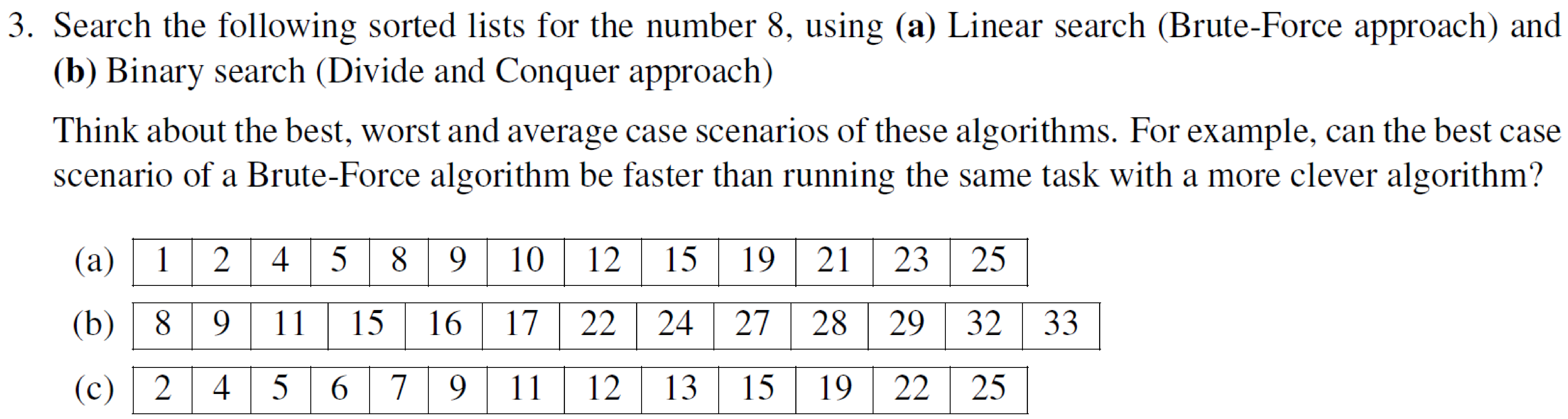
Heuristic
Approximate - uses knowledge about the possible answers to make a good guess

Simulation
Approximate - generates lots of possibilities to pick a good one

Divide and conquer
Exact - divide the problem into easier to solve subproblems

Exercise 3

Exercise 3 answer



3. Search the following sorted lists for the number 8, using **(a)** Linear search (Brute-Force approach) and **(b)** Binary search (Divide and Conquer approach)

   Think about the best, worst and average case scenarios of these algorithms. For example, can the best case scenario of a Brute-Force algorithm be faster than running the same task with a more clever algorithm?

(a)	1	2	4	5	8	9	10	12	15	19	21	23	25
(b)	8	9	11	15	16	17	22	24	27	28	29	32	33
(c)	2	4	5	6	7	9	11	12	13	15	19	22	25

CISSA

FOUNDATIONS OF COMPUTING COMP10001

```
SWOTVAC Terminal
Last login: The night before exam
(base) cissa@education ~ %
 ...
if not confident:
    attend(CISSA_REVISION_WORKSHOP)
else:
    attend(CISSA_REVISION_WORKSHOP)
 ...
print_info()
>>>
```

29 MAY 2023

11:00 AM – 1:00 PM

DUAL DELIVERY

ALAN GILBERT G18

SWOTVAC REVISION WORKSHOP 2023

UMSU | AFFILIATED CLUB

WORKSHOP

○ ○ ○ ○

Grok, problems from sheet, ask me questions :)

See you next week!