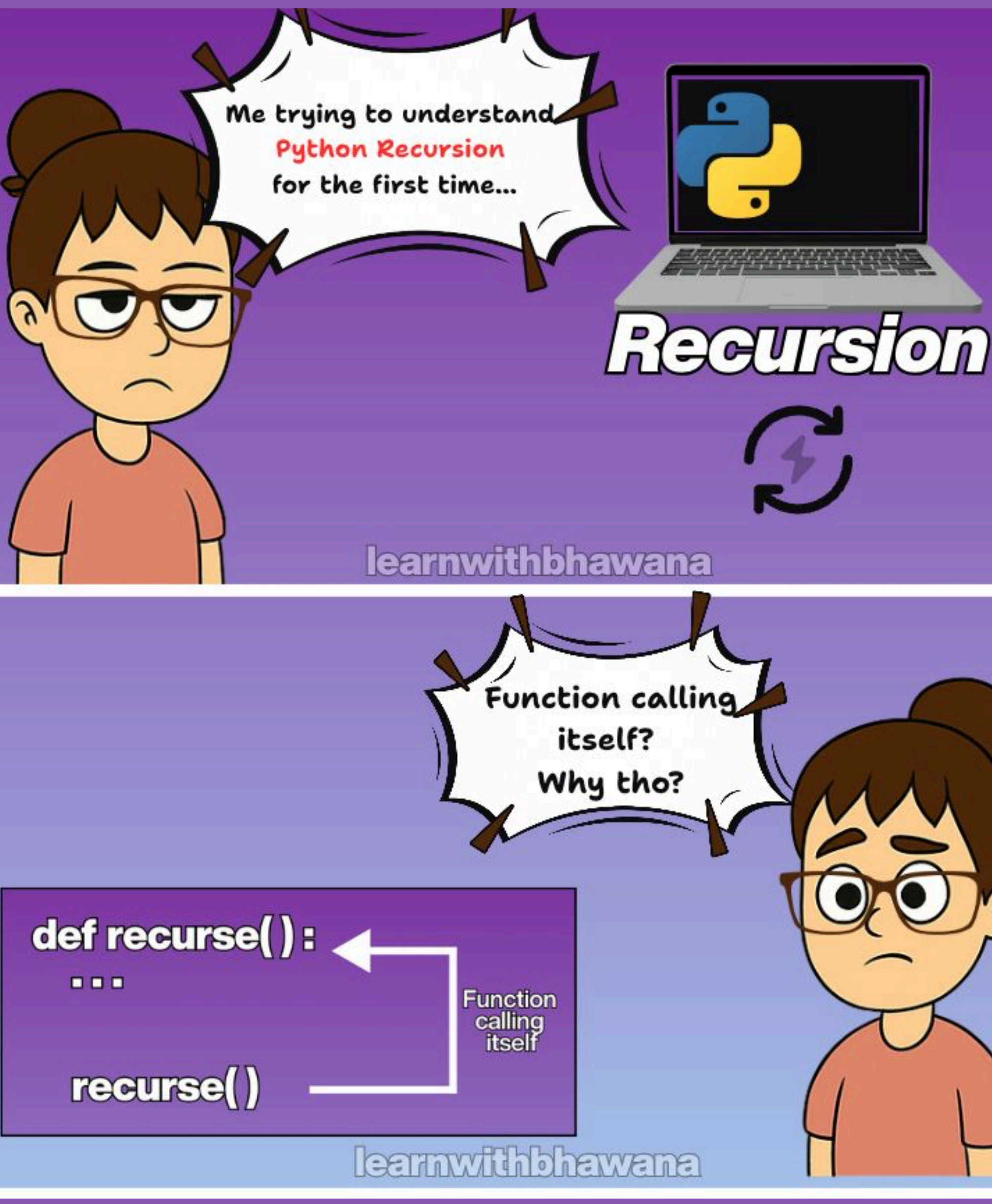
"Python Recursion & Me"

Adventure

Recursion

LearnwithBhawana

learnwithbhawana



learnwithbhawana

learnwithbhawana

recursion.py > ...

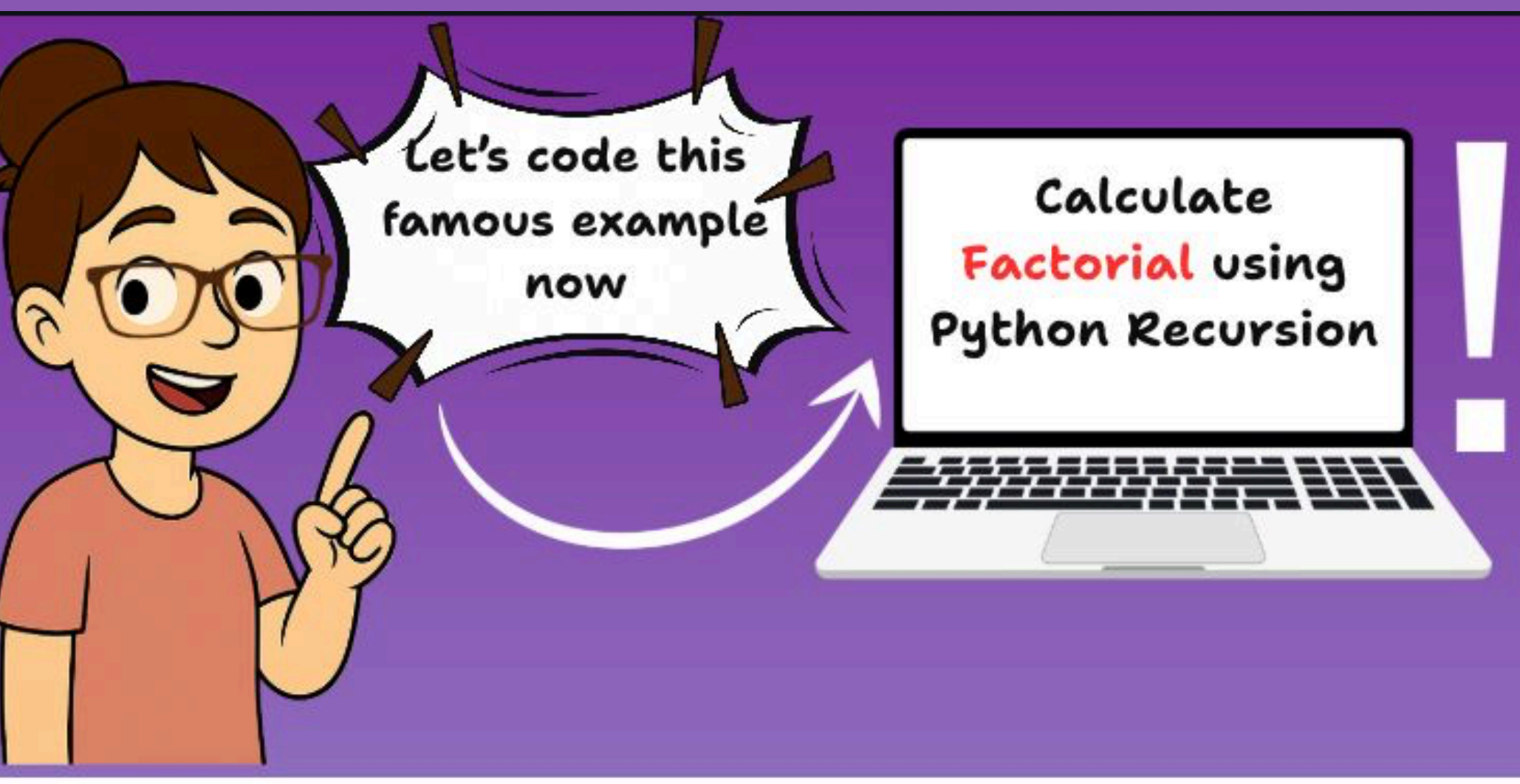
```python
def greet(x) :
    if x ==0:
        print("Stop")
    else:
        print("Hello")
        greet(x-1)

#--Function call --
greet(6)
```


Output:

```
Hello
Hello
Hello
Hello
Hello
Hello
Stop
```

learnwithbhawana



learnwithbhawana

```python
#Factorial using recursion
def factorial(x) :
    if x ==0:
        return 1
    else:
        return (x * factorial(x-1))

#--Function call --
print(factorial(3))
```

Output: 6

learnwithbhawana

- In this example, I call factorial(3) .
- It says: 3 × factorial(2)
- Which is: 3 × 2 × factorial(1)
- Which is: 3 × 2 × 1 × factorial(0) → returns 1

💥 Final Answer: 3 × 2 × 1 × 1 = 6

```python
#Factorial using recursion
def factorial(x) :
    if x ==0:
        return 1
    else:
        return (x * factorial(x-1))

#--Function call --
print(factorial(3))
```

Code Explanation

Output : 6

- So Recursion is like a loop
- But Instead of Repeating, it calls itself again and again until done.

Let's
dive a bit Deeper with
Fibonacci sequence

Let's GO

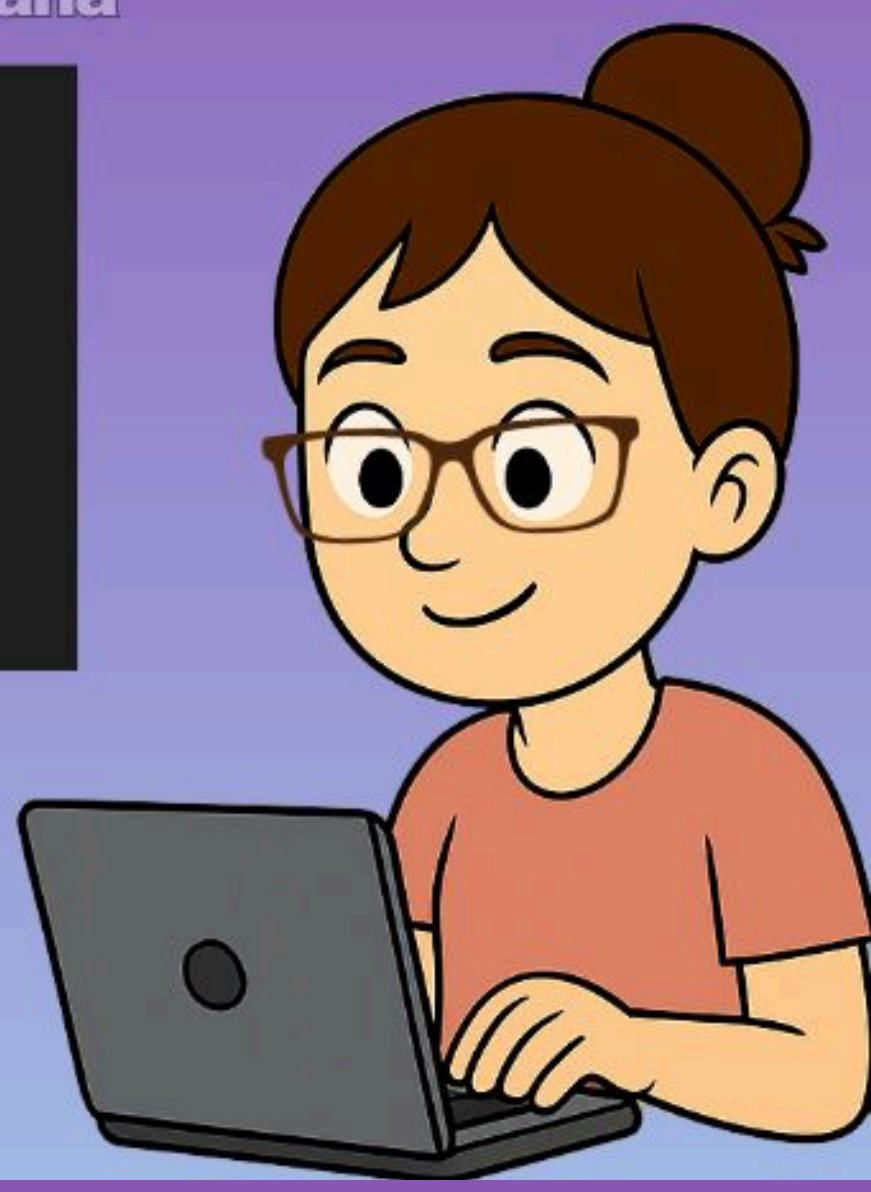
Fibonacci Sequence

- → 0+1 = 1
- → 1+1 = 2
- → 1+2 = 3
- → 2+3 = 5
- → 3+5 = 8
- → 5+8 = 13
- → ...

learnwithbhawana

```python
#Fibonacci using recursion
def fib(n):
    if n <= 1:
        return n
    return fib(n-1) + fib(n-2)

#--Function call --
print(fib(4)) #Output : 3
```



Output: 3

learnwithbhawana

Now my function is calling itself twice! Like a branching tree

Let's GO

Recursion Tree (for fib(4)):

```
fib(4)
├── fib(3)
│   ├── fib(2)
│   └── fib(1)
└── fib(2)
```

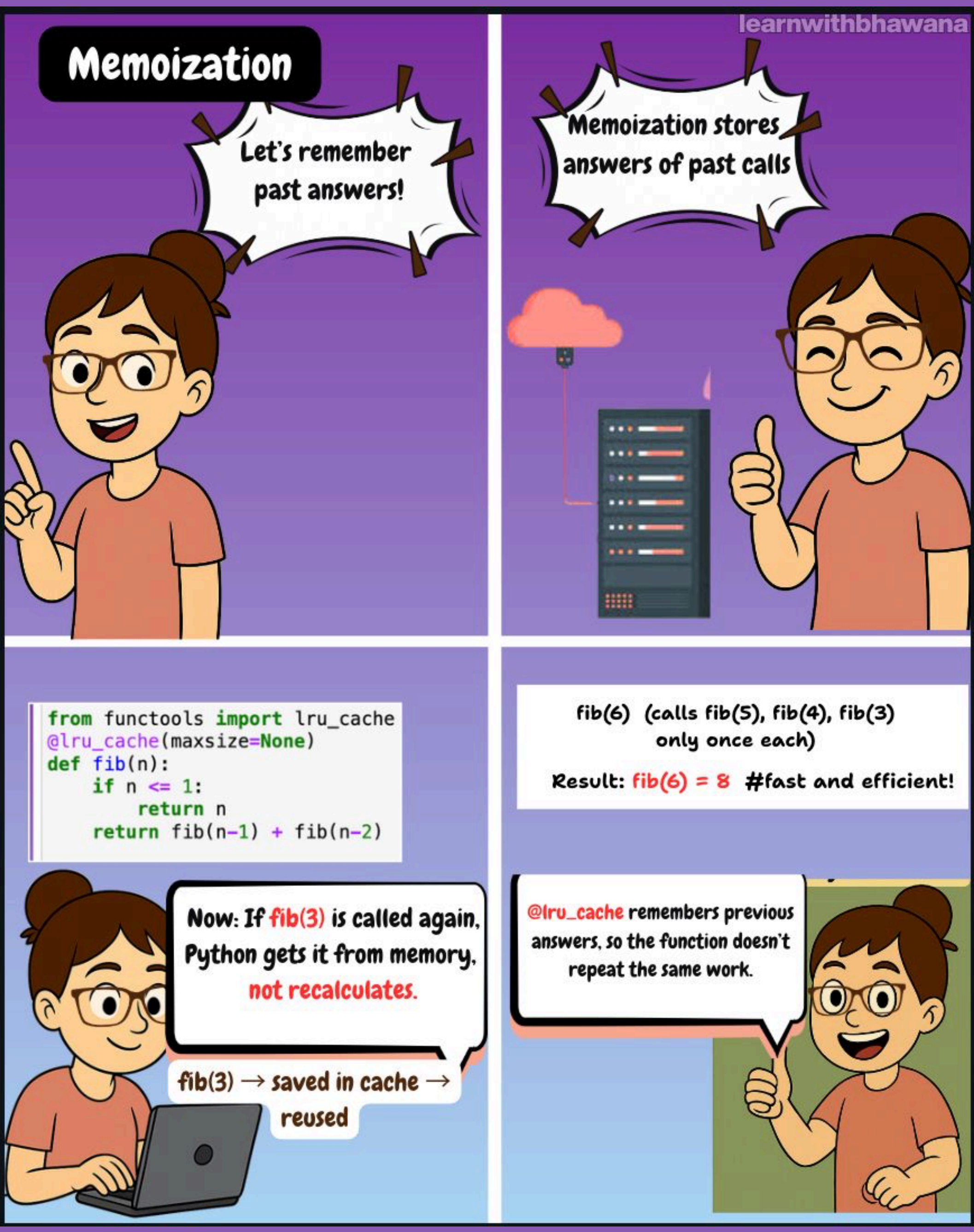

So, Recursion is **Powerful**.
Just remember these rules:

1. Always define a Base Case.
2. Function should call itself with a smaller input.
3. Trust the function to solve the smaller problem!

learnwithbhawana

Stack OverFlow risk


```
def recurse( ) :
    ...
    recurse( )
```

learnwithbhawana